MOSS ARC/NODE and DBMS
DESIGN DOCUMENT

FINAL DRAFT

by

AUTOMETRIC, Inc.

Carl Reed
Randy Frosh
Jim Kramer
Robert Starling

March, 1985

#1226525644

ID: 88076355

G
70.2
.R44
1985

Acknowledgements

Autometric wishes to acknowledge the input from Lyn Oleson, (USGS), Robin Fegeas (USGS), Dave Bankers (TGS), John Heasley (FWS), Jan Van Roessel, (TGS), Eric Strand (BLM), Dennis Colareli (TGS), Bruce Wright (USGS), Harry Niedzwiadek (Autometric), Bruce Blackwell (Autometric), and Lee Bender (Autometric) for their input and comments in the preparation of this document.

BLM Library
Denver Federal Center
Bldg. 50, OC-521
P.O. Box 25047
Denver, CO 80225

TABLE OF CONTENTS

1.0 Introduction.......... 1
1.1 Background.......... 1
1.2 Design Assumptions.......... 2
1.2.1 Phased approach for implementation.......... 2
1.2.2 Characteristics of Map Data Coming into MOSS.... 3
1.2.3 Topological Validity.......... 4
1.2.4 MOSS is Not a Digitizing System.......... 4
1.2.5 Projection Independence and Measurement Flexibility.......... 4
1.2.6 Accomodate all Data Types.......... 4
1.2.7 All Enhancements Will be Fully Documented....... 5
1.2.8 Current User Interface is Maintained.......... 5
1.2.9 On-line Help.......... 5
1.2.10 Continuity With Existing Data.......... 6
1.2.11 Integrated Terrain Units Not Required.......... 6
1.2.12 Strive for Reliability.......... 6
1.2.13 Handle Z Coordinates.......... 6
1.2.14 Enhance Modularity.......... 6
1.2.15 ANSI FORTRAN 77.......... 7
1.2.16 16/32 Bit Software Portability.......... 7
1.2.17 Data Set Transport.......... 7
1.2.18 A Map Sheet Based Database Structure.......... 8
1.3 Four Main Design Decisions.......... 8
1.3.1 Phased Implementation.......... 8
1.3.2 DBMS Interface.......... 8
1.3.3 Where to store the spatial data.......... 10
1.3.4 Map Sheet versus Tiling.......... 11
1.3.5 Multiple files versus one file.......... 13
1.3.6 Design Overview.......... 14
1.4 Goals and Objectives.......... 14
1.5 Summarization of Functions from January Report.......... 15
1.5.1 Data Entry.......... 15
1.5.2 Attribute/DBMS.......... 16
1.5.3 Retrieval.......... 16
1.5.4 Analysis.......... 17
1.5.5 Data Conversion.......... 17
1.5.6 Data Description.......... 18
1.6 Logical Processing Modules.......... 18
1.6.1 Attribute Manipulation.......... 18
1.6.2 Data Entry.......... 19
1.6.3 Verification/Polygon Formation.......... 20
1.6.4 Overlay.......... 20
1.6.5 Zone Generation.......... 21
1.6.6 Polygon to Arc.......... 21
1.6.7 Vector to Raster.......... 22
1.6.8 Additional Modules.......... 22
1.7 Overview.......... 22

2.0 Logical/Physical Data Structure.......... 23
2.1 Definitions.......... 23
2.1.1 Geometry - General.......... 24

2.1.2 Feature Geometry - Assumptions.................. 25
2.1.3 Geometry - Definitions........................... 26
2.1.4 Geometrical Attributes........................... 28
2.1.5 Attributes - General............................. 29
2.2 Logical Schema for Data Structure...................... 30
2.2.1 Node File.. 30
2.2.1.2 Relationship to Other Files.................. 31
2.2.2 Edge File.. 32
2.2.3 Coordinate File.................................. 33
2.2.4 Feature File..................................... 34
2.2.5 Feature Chain File............................... 35
2.2.6 Label File....................................... 36
2.2.7 Cartographic File................................ 36
2.2.8 Attribute File................................... 36
2.2.9 Other Files...................................... 37
2.3 Physical Data Structure................................ 37
2.3.1 File Organization................................ 38
2.3.2 Node File.. 39
2.3.3 Edge File.. 41
2.3.4 Coordinate File.................................. 42
2.3.5 Feature File..................................... 44
2.3.6 Feature Chain File............................... 46
2.3.7 Attribute File................................... 47
2.3.8 Cross Reference Files............................ 50
2.3.9 Fast Index File.................................. 50
2.3.10 Map Header File................................. 52
2.3.11 Map Name Conventions............................ 53
2.4 Summary.. 54

3.0 Logical Processing Modules - Detailed Discussion........... 55
3.1 Attribute Manipulation................................. 55
3.1.1 Attribute Reclassification....................... 55
3.1.1.1 Description.................................. 56
3.1.1.2 Reclassification Interface Information....... 56
3.1.1.3 Reclassification Flow Sequence............... 56
3.1.1.4 Attribute Reclassification................... 57
3.1.2 Attribute Entry/Validation....................... 57
3.1.2.1 Description.................................. 57
3.1.2.2 Interface Information........................ 58
3.1.2.3 Software Flow................................ 58
3.1.2.4 User Interface............................... 59
3.1.3 Attribute Update................................. 60
3.1.3.1 Description.................................. 60
3.1.4 Delete... 60
3.1.4.1 Delete Description........................... 60
3.1.4.2 Delete Interface Information................. 61
3.1.4.3 Delete Software Flow......................... 61
3.1.4.4 Delete User Interface........................ 62
3.1.5 Merge.. 62
3.1.5.1 Merge Description: Type I.................... 62
3.1.5.2 Interface Information........................ 63
3.1.5.3 Merge Software Flow.......................... 63
3.1.5.4 Merge User Interface......................... 64
3.1.5.5 Attribute Merge: Type II..................... 64
3.1.5.6 Interface Information........................ 64

3.1.5.7 Merge Software Flow........................ 65
3.1.5.8 Attribute Merge: Type III.................. 65
3.1.6 Attribte Dump to Interchange File............... 65
3.1.6.1 Description................................ 65
3.1.6.2 Dump Interface Information................. 65
3.1.6.3 Dump Software Flow......................... 66
3.1.6.4 Dump User Interface........................ 66
3.1.7 Attribute Browse................................ 66
3.1.7.1 Browse Description......................... 66
3.1.8 Attribute Report Generation..................... 66
3.1.8.1 Report Description......................... 66
3.1.8.2 User Interface (current environment)....... 67
3.1.9 Attribute Retrieval............................. 67
3.1.9.1 Attribute Retrieval Description............ 67
3.2 Data Entry.. 67
3.2.1 Geometric Shapes................................ 68
3.1.2.4 User Interface............................. 68
3.2.2 Keyboard Entry of Geographic Data............... 69
3.2.2.1 Description................................ 69
3.2.3 Edgematch between Adjacent Sheets............... 70
3.2.3.1 Description................................ 70
3.2.3.2 Edgematch Interface Information............ 70
3.2.3.3 Edgematch Software Flow.................... 70
3.3 Verification/Polygon Formation...................... 72
3.4 Arc/Node Overlay.................................... 73
3.4.1 Description..................................... 73
3.4.1.1 Overlay User Input Description............. 73
3.4.1.2 Interface Information - User processor for Overlay...................................... 74
3.4.1.3 Software Flow.............................. 74
3.4.1.4 User Interface............................. 75
3.4.2 Pre-process Edge Data........................... 76
3.4.2.1 Description................................ 76
3.4.2.2 Interface Information...................... 77
3.4.2.3 Software Flow.............................. 77
3.4.3 Intersection Processor.......................... 78
3.4.3.1 Description................................ 78
3.4.3.2 Interface Information...................... 79
3.4.3.3 Software Flow.............................. 80
3.4.4 Create new Arc/Node Map......................... 81
3.4.4.1 Description................................ 81
3.4.4.2 Interface Information...................... 81
3.4.4.3 Software Flow.............................. 81
3.4.5 Polygon Formation............................... 82
3.4.5.1 Description................................ 82
3.4.6 Attribute Integration........................... 82
3.4.6.1 Description................................ 82
3.4.6.2 Interface Information...................... 83
3.4.6.3 Software Flow.............................. 83
3.4.7 Sliver Removal.................................. 84
3.4.7.1 Description................................ 84
3.4.7.2 Interface Information...................... 85
3.4.7.3 Software Flow.............................. 85
3.4.8 Merge/Dissolve.................................. 86
3.4.8.1 Description................................ 86

3.4.8.2 Interface Information........................ 87
3.4.8.3 Software Flow................................ 88
3.4.8.4 User Interface............................... 88
3.5 Zone Generation...................................... 90
3.5.1 Description.. 90
3.5.2 Zone User Interface................................ 90
3.5.2.1 Description.................................. 90
3.5.2.2 Interface Information........................ 91
3.5.2.3 Software Flow................................ 91
3.5.2.4 User Interface............................... 92
3.5.3 Zone Generation Process............................ 93
3.5.3.1 Description.................................. 93
3.5.3.2 Interface Information........................ 95
3.5.3.3 Software Flow................................ 95
3.5.4 Overlay Processor.................................. 96
3.6 Polygon to Arc....................................... 96
3.6.1 Description.. 96
3.6.2 User Interface..................................... 96
3.6.2.1 Description.................................. 96
3.6.2.2 Interface Information........................ 96
3.6.2.3 Software Flow................................ 97
3.6.2.4 User Interface............................... 97
3.6.3 Create Candidate File.............................. 97
3.6.3.1 Description.................................. 97
3.6.3.2 Interface Information........................ 98
3.6.3.3 Software Flow................................ 98
3.6.4 Invoke Overlay Processor........................... 98
3.7 Vector to Raster..................................... 98
3.7.1 Description.. 98
3.8 Edge Intersection.................................... 99
3.8.1 Description.. 99
3.8.2 Interface Information..............................100
3.8.3 Software Flow......................................100
3.9 DBMS Shell...101
3.9.1 Description..101
3.9.2 Open Attribute Database............................102
3.9.2.1 Description..................................102
3.9.2.2 Interface Information........................102
3.9.2.3 Software Flow................................102
3.9.3 Validate and Find a Key............................103
3.9.3.1 Description..................................103
3.9.3.2 Interface Information........................103
3.9.3.3 Software Flow................................103
3.9.4 Get a Row of Values................................103
3.9.4.1 Description..................................103
3.9.4.2 Interface Information........................104
3.9.4.3 Software Flow................................104
3.9.5 Put a Row of Values into Attributes Database.....104
3.9.5.1 Description..................................104
3.9.6 Get a Value From a Row.............................104
3.9.6.1 Description..................................104
3.9.6.2 Interface Information........................105
3.9.6.3 Software Flow................................105
3.9.7 Put a Value into a Row of Attribute Data.........105
3.9.7.1 Description..................................105

        3.9.7.2 Interface Information....105
        3.9.7.3 Software Flow....106
    3.9.8 Delete a Row of Attribute Data....106
        3.9.8.1 Description....106
        3.9.8.2 Interface Information....106
        3.9.8.3 Software Flow....106
    3.9.9 Add a Row....107
        3.9.9.1 Description....107
    3.9.10 Change a Value in a Row....107
        3.9.10.1 Description....107
    3.9.11 Create a New Dataset....107
        3.9.11.1 Description....107
    3.9.12 Delete an Attribute Database....107
        3.9.12.1 Description....107
    3.9.13 Boolean Searches....107
        3.9.13.1 Description....108
        3.9.13.2 Interface Information....108
        3.9.13.3 Software Flow....108

4.0 Implementation....109
    4.1 Introduction....109
    4.2 Incorporating a New Data Type in MOSS. [Phase I]....109
    4.3 Attribute manipulation. [Phase I]....111
    4.4 Data Entry (Task 7)....118
        4.4.1 GENERATE Enhancement....118
        4.4.2 Edge Match....119
    4.5 Verification/Polygon Formation....119
    4.6 Arc/Node Overlay....120
        4.6.1 User Interface....120
        4.6.2 Edge Intersection....121
        4.6.3 Preprocessor....121
        4.6.4 Arc/Node to Polygon Formation....121
        4.6.5 Attribute Information....121
        4.6.6 Sliver Removal....121
    4.7 Zone Generation (Task 10)....121
        4.7.1 User Interface....122
        4.7.2 Zone generation Algorithm....122
        4.7.3 Invoke overlay processor....122
    4.8 Polygon to Arc....122
    4.9 Vector to Raster....123
    4.10 Miscellaneous Modules Required for Phase I Implementation (Task 13)....123
        4.10.1 AUDIT....123
        4.10.2 ADD....123
        4.10.3 DLG2MOSS....124
        4.10.4 DUMP....124
        4.10.5 SIZE....124
        4.10.6 STATISTICS (DESCRIBE)....124
        4.10.7 STATUS....124
        4.10.8 Merge/Dissolve....125
    4.11 Phase I Summary....125
    4.12 Phase II....125
    4.13 Phase III....127

5.0 Level of Effort..129
    5.1 Introduction...129
    5.2 Phase II...131
    5.3 Phase III..132

APPENDICES:

APPENDIX A
APPENDIX B
APPENDIX C
APPENDIX E
APPENDIX F
APPENDIX G

1.0 INTRODUCTION

This draft report is submitted as a description of the detailed design specification for implementing an arc/node data structure in MOSS. Additionally, this report addresses the techniques for interfacing relational database management systems with MOSS. The purpose of this draft is to provide the opportunity for interested technical personnel to review the design and to make comments that can be incorporated into the final design document. Please note that the design focuses on those high priority functions defined in the Functional Requirements Document.

1.1 Background

This report is the second deliverable under Phase II of the Feasibility and Design Study for the Enhancement of MOSS. Phase I of this study emphasized the functional requirements for an enhanced version of MOSS. Under Phase I, both the users of MOSS and the literature were interrogated to develop a prioritized list of enhanced functional capabilities in the software. The emphasis of the Phase I effort was on those functions most closely related to an arc/node, or topological, data structure and attribute storage, and manipulation. While there are many areas for improvement in MOSS, these two were deemed the most important. The results of the functional specifications study may be found in a document dated January 1985.

As with Phase I, Phase II will focus on the arc/node data structure (and those analytical functions most heavily impacted by such a data structure) and on the enhancement of the attribute manipulation capabilities, specifically stressing the interface between MOSS and a relational DBMS.

1.2 Design Assumptions

There are a number of design assumptions that constrain the boundaries of the problem domain. These assumptions, many of which were originally defined in the Functional Specifications Document (January 1985) have been discussed at length both by ourselves and the design review committee. These assumptions are described.

1.2.1 Phased approach for implementation:

One of the big questions has been "How best to implement the arc/node enhancements and DBMS interface for MOSS?" There are two possible scenarios that have any validity. The first scenario would be to implement an entirely new system external to MOSS. Users would continue to use the old MOSS until the new system became available. They could then switch to the new system. The second approach is to modify and enhance the existing version of MOSS over a much shorter period of time to provide the users access to the high priority functional requirements identified in the January report. Once these enhancements are completed then Phase 2 development would begin. The medium priority functions would be implemented and tested during Phase 2. The final phase would be to completely replace the polygon processing in MOSS with arc/node processing. We have then arrived at the same logical evolution point as in Scenario one.

For the following reasons, Scenario two is considered to be the overall optimal implementation strategy:

- The first scenario would require a 2 to 3 year startup time. Users require certain functions, such as polygon overlay, in a much shorter time frame than that.
- In a production environment users cannot afford any down time. Scenario two would allow a new data type to be

"slipped into" MOSS with minimal perturbation to MOSS - the ADD command would look slightly different and ADDWAMS would disappear!

- Users could decide when they wish to switch to using arc/node maps and not be driven by the technocrats.
- The 50,000 or so existing MOSS maps would be usable without having to convert them immediately to the arc/node structure.
- Funding for software development is often allocated in a piecemeal fashion. Scenario two accomodates the funding process more accurately.

The actual phasing is described in much greater detail in Section 4 - Implementation Strategy.

1.2.2 Characteristics of Map Data Coming into MOSS

We assume that the data coming into MOSS from an outside digitizing source, such as AMS, ADS, or the USGS DLG, is:

- Not "spagetti" but is comprised of arcs, nodes, and coordinates. Feature representations are not necessary;
- In a common coordinate or projection system,
- Tied at the edges between adjacent edges; i.e. edge match has been performed. However, this will not always be the case especially for older data. Therefore, arc/node MOSS will support an edge match procedure.

We do not assume that the data is topologically valid. The verification process will identify problems in topology. Since an AMS to DLG3 convertor and an ARC/INFO to DLG3 convertor already exists, we recommend that the DLG3 ASCII format be adopted for the interchange format for arc/node MOSS.

1.2.3 Topological Validity

We assume that the arc/node version of MOSS will maintain topological validity. This means that the topological verification processor must recognize and report errors in topology such as loops, spikes, missing arcs, and gaps as well as perform feature building and island assignment.

1.2.4 MOSS is Not a Digitizing System

MOSS is not and will not be a map or photo digitizing system. However, such operations as boundary match, geometric figure generation, and keyboard coordinate entry will be accommodated. Also, the requirements for data entry must be supported in the data structure definition.

1.2.5 Projection Independence and Measurement Flexibility

Both the data structure and the user interface must be able to accommodate different types of units of measure. For coordinate data, units of measure could be feet, meters, inches, or decimal degrees. Conversely, regardless of the units of measure of storage, the user must be able to specify input parameters in any unit of measure. They should also be able to specify units of measure for output, such as in MOSS area tables.

1.2.6 Accomodate all Data Types

The data structure in MOSS must still be able to accommodate both vector and raster structures. Further, the processing modules should protect the user from data types. A current example is PLOT. The PLOT command works on any data type (point, line, area, DEM, discrete raster, text, and so on). The user does not have to do anything special to handle different data types. As modules are changed to the new arc/node structure, this data type independence must be implemented.

1.2.7 All Enhancements Will be Fully Documented

Any new software or any modified software related to implementing the design specified in this document will be fully documented. This documentation will include:

- Software Change reports,
- Subroutine headers,
- In line documentation,
- Command helps,
- Helps for any software prompt to the user,
- User manual upgrades,
- Training Manuals.

1.2.8 Current User Interface is Maintained

In order to maintain continuity during the implementation stage, the current user interface will be maintained in MOSS. However, as any existing command module is upgraded to reflect the new data structure and DBMS interface, all user interaction will be stripped out of the MOSS command processor for that command and put into a new subroutine. This new subroutine will pass back the necessary parameters to the processor. In this way, future changes to the user interface will be facilitated. Recently, this re-modularization has been done to PLOT and QUERY.

1.2.9 On-line Help

In addition to the current help files, any new software that requires a user interface will allow the user to issue a ?HELP at any prompt. Information related to that prompt will then be given to the user.

1.2.10 Continuity With Existing Data

The phased approach will allow the user to process both full polygon and arc/node data at the same time.

1.2.11 Integrated Terrain Units Not Required

The user will have total control over mixing and matching layers of map data. They will not be required to overlay all data into a single layer comprised of integrated terrain units in order to build the database.

1.2.12 Strive for Reliability

In order to insure that the data processing capabilities of the enhanced MOSS are reliable, we propose that a rigorous testing procedure be pursued. The procedure should include:

- A standard test data set with known pathological cases;
- Module testing,
- Routine testing,
- System testing,
- Beta site testing,
- Production testing.

1.2.13 Handle Z Coordinates

The data structure must allow for storing not only (x,y) location but (x,y,z) locations for both nodes and coordinate data.

1.2.14 Enhance Modularity

One of the primary functions of this design document is to describe processing modules in enough detail so that any new developments related to arc/node or DBMS interface will be modular in their implemention.

1.2.15 ANSI FORTRAN 77

As much as possible, ANSI FORTRAN 77 will be used for all new code and all changes to existing code. This should be pursued to help insure ease of portability between CPU's. It also presupposes that the existing code has been converted to FORTRAN 77 on all existing hardware configurations.

1.2.16 16/32 Bit Software Portability

Since there will be 16 bit hardware configurations around for some time, the software must be operant on both 16 bit and 32 bit environments. FORTRAN 77 will allow this. By using default variable/array declarations in FORTRAN 77, this can be accomplished. Further, all new software should be parameter initialized in one block data. Word size, record sizes, and other system specific information can be stored in common and be used by certain primitives, such as character manipulation, to aid in portability.

1.2.17 Data Set Transport

There must be the ability to move 32 bit structured MOSS data sets into the 16 bit environment. This will be accomplished through the use of a reformatting program that moves 16 bit data sets into the 32 bit realm and visa-versa. This software must check for potential addressing problems, values that are too large for the 16 bit environment and so forth. If these problems exist, then the map sheet will have to be passed into smaller manageable units. This form of data transport is required because 32 bit addressable data sets simply are unsupportable in the 16 bit environment.

1.2.18 A Map Sheet Based Database Structure

Currently in MOSS, the database is parsed into map sheets. All retrieval and manipulation is on a sheet by sheet basis. We recommend that Phase I enhancement of MOSS continue to use the map sheet approach. However, the final implementation should be tile based. The design must consider the tiled approach to map data storage. The discussion of tiles is continued in section 1.3.

1.3 Four Main Design Decisions

The previous section described the design assumptions for arc/node design effort. This section describes four major design decisions related to the assumptions, the design of the system, and to the functional specifications presented in the January document. These are major design decisions and impact discussions in the remainder of the document.

1.3.1 Phased Implementation

The decision for a phased implementation, which has become a design assumption and described in section 1.2.1 is elaborated in Section 4. No further discussion is given here.

1.3.2 DBMS Interface

A key decision point was where to draw the interface boundary between MOSS and a relational DBMS. This decision had to be made in order to determine what the impact would be on existing software as well as what new software would have to be defined and implemented.

The interface boundary may be thought of as a continuum. At one end there would be total integration of MOSS with a relational DBMS. In this case all the spatial data as well as the attribute

data would be handled by the RDBMS. MOSS could not run without the RDBMS. MOSS would require major modification and all users would be forced to use the designated RDBMS. This simply is not realistic. Some users will not even want a RDBMS. Others will not have a large enough CPU to accomodate such an implementation. All users would be at the mercy of the vendor or group supporting the RDBMS: the RDBMS software is subject to change without notification. Portability is stifled. Another solution is required.

At the other extreme on the continuum is the use of auxiliary files to interchange "keys" between MOSS and the RDBMS. In this scenario, the user performs attribute searches in the RDBMS. A disk file of Feature I.D.'s and map names is written to disk. This intermediate disk file is then used by MOSS to retrieve features from the database. Such an approach, while simple to implement, does not give the MOSS user direct access to the attribute database. Also, functions such as overlay become cumbersome for user's to operate if they must constantly go back and forth between MOSS and the RDBMS. Therefore, this interface strategy is not a viable long term approach.

In order to allow users to directly access the DBMS but not tie them to any one system as well as to continue to allow users to use the current multiple attributes capability, we decided that an interface between the continuum extremes is desirable. In such an environment, MOSS is not dependent on any one DBMS. Instead, a DBMS Interface Shell is used. This shell contains a standardized set of subroutine interface calls that MOSS application modules use. These routines control all the interface operations with the DBMS , regardless of whether it is RIM, INGRES, or the current Multiple Attributes capability. Both high frequency and low frequency transactions are supported. Section 3 describes the DBMS shell and the interfaces between MOSS and the attributes file. Figure 1.1 pictorially represents the continuum.

1.3.3 Where to store the spatial data

There has been a lot of discussion (see Footnote below) about whether the spatial structure/data should be implemented entirely within a relational DBMS or should be kept external from it. From our point of view, to completely "enclose" the spatial

---

Footnote 1

Antonacci F., et al.: AQL: A Relational Database Management System and its Geographical Applications. In: Blaser A. (ed.): Data Base Techniques for Pictorial Applications. Springer Verlag, 1979, pp. 569-599.

Becerril J.L., Casajuana R., Valer F.: Representation, Handling and Storage of Geographical Information Supported by a relational Database. URBAN DATA MANAGEMENT 9th European Symposium, Valencia (Spain), October 1982, pp. 1-22.

Cox N.J., Aldred B.K., Rhind D.W.: A Relational Data Base System and a Proposal for a Geographical Data Type. Geo-Processing, 1(1980),, Elsevier Scientific Publ. Co., Amsterdam 1980, pp. 217-229.

Haskin R.L., Lorie R.A.: On Extending the Functions of a Relational Database System. ACM SIGMOD Intl. Conf. on Management of Data, June 1982, pp. 207-212.

Lorie, R. and Meire, A.,: Using a Relational DBMS for Geographical Databases. In Geoprocessing 2 (1984) Elsevier Scientific Publ. Co., Amsterdam.

Meier A., Lorie R.A.: Implicit Hierarchical Joins for Complex Objects. IBM Research Report RJ3775, January 1983.

structure in an RDBMS would require a total interface to MOSS. The same arguments stated in 1.3.2 become applicable. Also, to our knowledge, no production GIS is currently using this approach. Thus, to use a total interface would put us into the realm of "untested technology". We propose keeping the spatial structure separate from and external to the RDBMS.

Nevertheless, separation of spatial data from the RDBMS does not preclude pursuing the use of logical pointers instead of physical pointers in the database structure definition. If this philosophy is adopted then each of the files defined in Section 2 may be thought of as tables (relations) in the RDBMS. If, at some future point, the relational technology is proven for storing spatial data, then the MOSS map files could easily slip into a RDBMS.

1.3.4 Map Sheet versus Tiling

Currently, MOSS uses the map sheet as the basic unit of storage. If all the sheets are the same size and shape, such as 7.5 minute quadrangles, they could be thought of as tiles. Tiles are nothing more than regular units of storage or access for map data in a large map database. As with the DBMS interface, considerable discussion was given to how map data should be logically and/or physically stored in the database. Again, there are a number of possible strategies.

One strategy is to not use any tiles or map sheets. In this scenario, all data is stored in one huge map file representing the entire project area. Data, such as arcs or features, are addressed through a spatial index file to ensure efficiency of access in a retrieval or analysis environment. The index changes as the density of data changes. Consider the "clean" tile in Figure 1.2.a. Only a simple index is required. However, if the density of features increases, such as through digitizing more data, then the indexing is going to have to change to ensure

processing efficiency. This is done by spatially parsing the file (based on density) such as in 1.2.b. This subdivision of the index continues as the density changes. The critical decision is how the index file is organized. B-trees, Quad-trees, exchange hashing, and Morton sequencing have been considered (see footnote 2). However, all the work for current vector based implementations of this approach are prototype only. No production systems exist. Problems of update, archival and so forth have not been properly addressed, Thus, we cannot use a developing, untested technology. However, the data structure design cannot preclude using such a technique at some later date.

The second approach is the use of regular tiles. This is essentially the concept used in AMS. Where AMS falls short is that it has forced the users to digitize from map sheets the same size as the tiles. Tiles are a very logical, easy to comprehend technique for parsing spatial data into the database. Retrieval efficiency becomes a function of tile size and number of features per tile. Tiling appears to be the way in which spatial database parsing is evolving (see Footnote 3). Both regular and dynamic tiling schemes are being investigated. Some systems are already using a limited implementation of tiling in production

---

Footnote 2

Mark, D. and Lauzen, J.P., Linear Quadtrees for Geographic Information systems. 1984.

Tammiren, M., Efficient Geometric Access to Multirepresentation Geo-Database. 1984. Geo-Processing 2, Elsevier Science Publishers, Co., Amsterdam.

Footnote 3

Work Autometric is involved with at this time for a variety of other customers, but no published reports are yet available.

environments. However, there are unresolved problems in tile size definition, re-tiling of the database, transaction analysis, ease of update, indexing of features across tile boundaries, and handling different tile sizes for different layers of information in the same database.

The third approach is to keep the current map sheet approach as is currently used in MOSS but remain cognizant of the fact that other storage techniques are being studied. Thus, the user should not be kept from being able to merge sheets together into a larger spatial grouping. The only missing entity will be the spatial index file discussed above. We are also not precluding tiling at some later date. If the map sheets are thought of as tiles, then the tiling structure could be implemented on top of the sheet (based) data structure described in section 2. For the above stated reasons, we will pursue the map sheet approach of map storage.

1.3.5 Multiple files versus one file

The final major decision was to determine whether for a given sheet (tile) all the spatial data files should be stored in one physical file on disk or a number of files on disk. After much discussion, we have opted for multiple files for the following reasons:

- Ease of update,
- Ease of garbage collection,
- Logical consistency,
- RDBMS requirements (future) for spatial storage
- Requirements for tiling (in the future).

The only drawback is that multiple files must be open at any one time. This drawback can be minimized by using one set of disk handler routines that handle all the logical/physical/file type differences.

1.3.6 Design Overview

To summarize, the overall design assumes:

- Map sheet implementation for spatial parsing and file management,
- Multiple files per sheet,
- DBMS shell,
- Logical pointers instead of embedded pointers in data structure,
- Phased implementation using existing MOSS instead of developing new system from scratch,
- Prototyped implementation strategy.

The next section summarizes the objectives for the remainder of this document

1.4 Goals and Objectives

The specific goal of Phase II in this study is to develop a scenario for actually implementing an arc/node structure and relational DBMS interface in MOSS. To achieve this goal, there are a number of objectives that must be met:

- Lay out the actual arc/node data structure in terms of required files and the formats of these files.

- Provide a description of the actual database primitive subroutines that will interface the applications software with this proposed data structure. The interface between existing MOSS code and this DBMS shell must be defined.

- Scope and describe how the key analytical functions, such as overlay, are impacted and how they could be upgraded to make best use of this new arc/node data structure.

- Describe at what levels a relational DBMS can be interfaced into MOSS and to describe what interface routines will have to be developed.

- Describe how a relational DBMS interface will impact the analytical routines, such as overlay.

- Develop a scenario for a phased implementation of these new enhancements. This scenario will include proposed level of effort.

- Identify analytical algorithms and techniques that are required for utilization of the arc/node data structure.

1.5 Summarization of Functions from January Report

This section summarizes the high priority functional requirements documented in the January 1985 document. The summary will provide the lead in for section 1.6 which groups these functions by logical processing function. Grouping is required in order to present a complete description of the interfaces and algorithms necessary to implement the functions in the new arc/node/DBMS environment.

1.5.1 Data Entry

- Produce geometric shapes of user defined dimensions, such as arcs, rectangles, and circles;
- Input coordinates by keyboard in any projection or rectilinear coordinate system;
- Provide manual edge match capability using CRT;
- Input from cross hairs, touch screen, light pen or mouse;
- Topological verification and validation;
- Accept USGS DLG 3 and DLG 3 (modified) data.

1.5.2 Attribute/DBMS

- The ability to dump out attributes stored in a MOSS database to an external file;
- The ability to accept attribute data from a variety of sources;
- Provide catalogue functions that the user can envoke against the attribute database;
- At a minimum, be able to support 100 attributes per feature;
- Validate attribute data as it's being entered into the database;
- Be able to retrieve using either simple (one field) or complex (many fields) boolean criteria;
- Provide basic attribute report generation, including a set of defaults, such as area and length reports;
- Allow users to merge attribute files;
- Allow users to delete attributes;
- Allow users to edit a value for a field;
- Queries should be able to be expressed in some common format, not as they are now;
- Create new fields based on algebraic manipulations on existing fields.

1.5.3 Retrieval

- Retrieve a set of features based on distance and/or an attribute search;
- The ability to retrieve data based on a geographic window in any projection or coordinate system;
- The ability to select based on any single or multiple attribute including map, arc and node headers;
- Permit boolean operands for attribute searches (AND, OR, NOT);
- Permit logical operands (LE, EQ, NE, GE, LT, GT);
- The ability to use one retrieval query to search across multiple map modules.

1.5.4 Analysis

- Generate zones about points, lines, or areas;
- Specify any units of measure for the width of a zone;
- Support overlay of all combinations of point, line and polygon data types;
- Provide AND, OR and NOT operations, including keeping or dropping interior boundaries in the OR operations;
- Keep track of the attributes through the overlay process;
- Give the user control over the attributes to be passed;
- Pre-process the spatial data before the overlay to increase efficiency. Such techniques as banding, ancillary fast files (such as in MOSS now), and B-trees should be explored;
- Store spatial knowledge in the database to increase overlay efficiency;
- Handle the basic sliver problem by allowing the user to keep slivers or eliminate them;
- Allow the user to specify a complex overlay query, such as (MAP(A).AND.MAP(B).AND.MAP(C)).OR.MAPD;
- Make the software modular so that it can also be used as a topological verifier for bringing external data into the system;
- Resolve islands.

1.5.5 Data Conversion

- Vector to raster conversions;
- Merge 2-D vector and DEM data sets to produce 3-D vector data sets;
- Reclassify attributes;
    - in a dataset or data base wide fashion based on transformation tables;
    - on a feature specific basis by substitution;
    - on an individual feature basis in an interactive mode or through the use of search and replace methods.
- Change resolution;

  - sliver removal after overlay type processes.
- Convert existing polygon files to arc/node data structure.

1.5.6 Data Description

- Lengths of features in a network;
- Perimeters for area features;
- Areas for area features;
- Frequency;
- Density;
- Report all the available information at a specific location;
- Describe increments along a feature and retain them as tick marks (which can be assigned a symbology). For example what vegetation type occurs every 100 meters along one side and/or the other, of a river bank;
- Describe (identify) the features within a specified distance of a feature.

1.6 Logical Processing Modules

When the list of high priority processing modules is considered, they can be easily regrouped by function into logical processing modules. These modules are described.

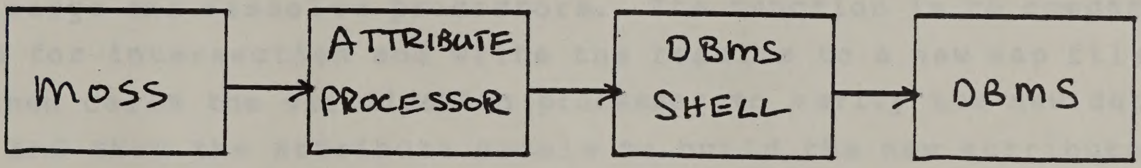
1.6.1 Attribute Manipulation

Attribute manipulation is the process of storing, retrieving, updating, and merging attribute information. In terms of processing functions, it must support:

- Reclassification
- Entry, including validation
- Update
- Delete
- Merge
- Dump to interchange file

- Browse
- Report generation
- Retrieval, including use of Boolean operators
- Common query format
- Compute operations

Notice that this list is derived from 1.5.1 through 1.5.6. This list can now be thought of as the attributes processing module. Any attribute manipulation would interface with this module. This module itself would interface with the DBMS shell.



1.6.2 Data Entry

While map data entry itself is not a high priority function, certain types of graphical input must be supported. These include:

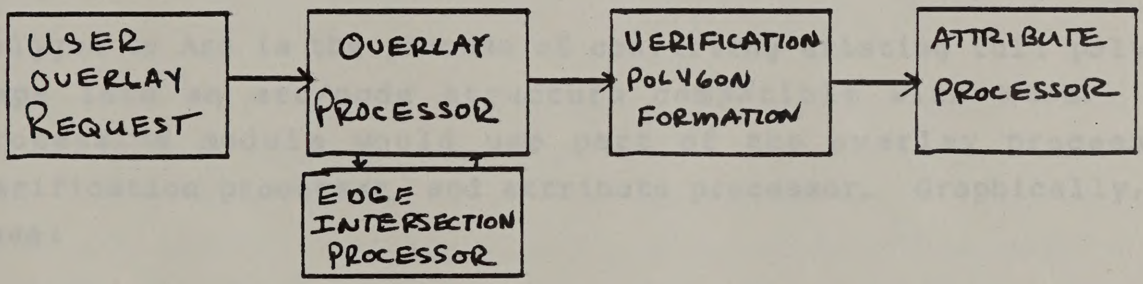
- Create and store geometric shapes, such as circles, rectangles, and geometric arcs.
- Key entry of coordinates describing a feature.
- Provide manual/machine edgematch between adjacent map sheets.
- Cross hair input to specify geographic location.

1.6.3 Verification/Polygon Formation

This module takes a geometric representation comprised of nodes and edges and validates its topology. As part of this process, it also performs a polygon formation process in which the edges are put together to form features. This particular module is required for validating data brought in from external data sources, verifying the results of the overlay processor and so forth.

1.6.4 Overlay

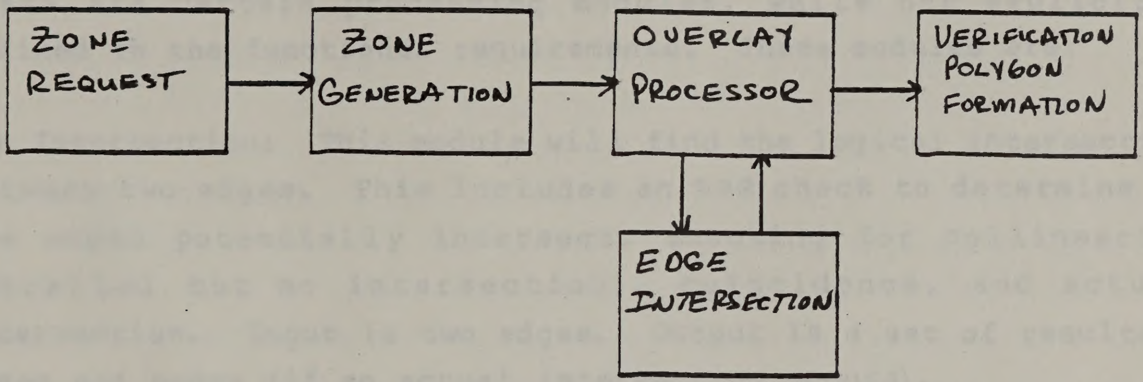
This processor interacts with the map overlay, polygon to arc, and merge and dissolve processors. Its function is to compare arcs for intersection and write the results to a new map file. It then calls the verification processor to verify the new data set and then the attribute module to build the new attributes file. Graphically:


USER OVERLAY REQUEST
OVERLAY PROCESSOR
VERIFICATION POLYGON FORMATION
ATTRIBUTE PROCESSOR
EDGE INTERSECTION PROCESSOR

Notice that the overlay processor calls an Edge intersection routine. This module takes two edges (arcs) and intersects them.

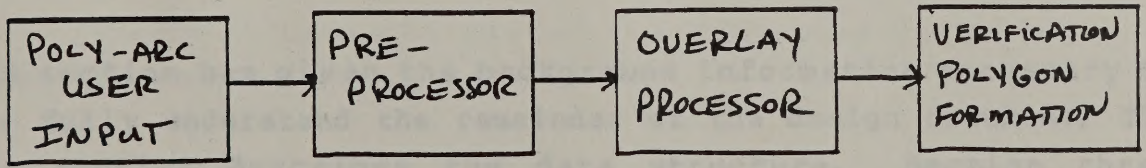
1.6.5 Zone Generation

Zone generation is the process of taking a feature and computing a zone of influence around it. These zones are also known as buffers. This processor requires the overlay processor to resolve overlapping zones. Graphically:


ZONE REQUEST
ZONE GENERATION
OVERLAY PROCESSOR
VERIFICATION POLYGON FORMATION
EDGE INTERSECTION

1.6.6 Polygon to Arc

Polygon to Arc is the process of converting existing full polygon maps into an arc/node structure compatible with MOSS. The processing module would use part of the overlay processor, verification processor, and attribute processor. Graphically, we have:


POLY-ARC USER INPUT
PRE-PROCESSOR
OVERLAY PROCESSOR
VERIFICATION POLYGON FORMATION

1.6.7 Vector to Raster

This module would convert arc/node maps into a raster format. Current processing modules would require minimal modification to support this capability.

1.6.8 Additional Modules

There are certain processing modules, while not explicitly defined in the functional requirements. These modules are:

Arc Intersection: This module will find the logical intersection between two edges. This includes an MBR check to determine if the edges potentially intersect, checking for collinearity (parallel but no intersection), coincidence, and actual intersection. Input is two edges. Output is a set of resultant edges and nodes (if an actual intersection occurs).

DBMS Shell: This module is a set of routines for interfacing MOSS to either its internal DBMS or to an external relational DBMS.

Database Primitives: This module is a set of subroutines for accessing the arc/node data structure. These routines would get and put records (node, arc, feature pointer) to the database. They are the key for developing new applications modules or upgrading existing modules.

1.7 Overview

This section has given the background information necessary to more fully understand the remainder of the design document. The next section describes the data structure. Section three describes the modules in much greater detail and provides algorithmic approaches.

2.0 Logical/Physical Data Structure

The section describes both the logical and physical data structures for the enhanced MOSS. Logical data structures refer to how the various parts of the structure relate to one another as well as the logical schema, or layout, of each of the files. Physical data structures refer to how the data is actually stored on disk.

The following data structure design takes into consideration the requirements for meeting the processing needs of the logical processing modules described in Section 1. Additionally, the AMS, ADS, ARC/INFO, GIRAS, ODDESSY, and MC & G data structures were reviewed to see how topological/spatial data structures have been implemented in other environments. Interestingly enough, the majority of these systems seem to have the same set of files, although there is major variation in content. Also, most existing systems use physical record pointers instead of logical key pointers. The use of logical pointers requires additional files to implement the proposed structure in environments in which the topological/spatial data is not being stored in a relational DBMS.

2.1 Definitions

Before a logical view of the structure is presented, certain definitions must be given in order to prevent unnecessary confusion.

We are concerned with two subsets of data when considering a geographic feature:

a. Geometric - the position of the feature in space.

b. Descriptive - the attributes/properties/ characteristics of the feature.

2.1.1 Geometry - General

There are two aspects of the geometry of figures in which we are interested - spatial and topological.

Spatial. The locations and configurations (Point, Line, Area) of figures as these are the forms in which most products are at present required, such as, conventional maps.

Topological. Topology may be defined as the study of the properties of geometric figures that remain unchanged even when such figures are distorted. That is, topology is not concerned with magnitudes or the projective properties of figures, but rather with relations that are invariant under distortion. Within a cartographic environment we are (except possibly as a function of coordinate conversion) concerned with the relationships of those figures insofar as they will assist in more efficient utilization of the geometric data held. Some examples of this are:

a. To ensure that the geometric data held is consistent within itself.

b. To assist in identifying relationships (coincidence, adjacency) in order to select data more rapidly.

c. To minimize data redundancy by only holding a coincident alignment once, once the coincidence has been identified.

d. To assist transversal through the geometric data.

2.1.2 Feature Geometry - Assumptions

For the purpose of discussion it is assumed that a feature is an entity located in space, any description being considered an attribute. For any feature it is assumed that the positional data defining that feature is that string of coordinates which when brought into the correct coordinate system will best define the feature boundary at the required resolution.

It is also assumed that the data positions are mapped in two-dimensions (x,y) onto a plane surface, with the positional values recorded to a specific resolution and referenced to a specified origin. Further it is assumed that the elevation (z) of the ground level at the point being mapped onto the plane will be held as an attribute of the coordinates at that point. The vertical dimensions of the feature with respect to ground level (z', z'') will be held as attributes of the feature if appropriate.

We have now defined a feature as being what is, in cartographic terms, an area feature. We can also define subsets of area features by adopting the convention of line feature and point features from conventional cartography. Mathematically point and line features are only subsets of area features. For example, there could be a "maximum dimension criteria" defined which, if exceeded, would force that feature subset into its containing set. This is also known as sliver removal or feature generalization.

We can now introduce the following definitions for feature type:

a. Area Feature. A two dimensional representation of an entity, with its perimeter being mapped onto the plane and with its elevation(s) relative to ground level being held as attributes.

b. Line Feature. A one dimensional representation of an entity, with its centerline being mapped onto the plane and with its width(s) and elevation(s) relative to ground level being held as attributes.

c. Point Feature. A zero dimensional representation of an entity, with its position being mapped onto the plane and with its dimensions being held as attributes.

2.1.3 Geometry - Definitions

The primitive geometric constructs from which all other geometric figures are derived are the point (or coordinate) and the straight-line segment, where[*]:

<Coordinate>::=[The implicit or explicit geographic position of a point on a plane or in a Volume.]

<Straight-Line Segment> ::= <Coordinate><Redundant Coordinate Set><Coordinate>

<Redundant Coordinate Set> ::= [the set of intermediate coordinate locations within a Straight-Line Segment, whose position may be calculated if necessary.]

<Segment Subset> ::= <Coordinate><Redundant Coordinate Set><Segment Subset> OR <null>

<Segment> ::= <Coordinate><Redundant Coordinate Set><Segment Subset><Coordinate>

OR ::= [one or more Straight-Line Segments, with the duplicated intermediate coordinates recorded only once]

---

[*] This is BNF notation

These are topologic definitions based on the geometric definitions.

<Node> ::= <Coordinate>

OR ::= [a Coordinate which is significant for one or more of the following reasons:

a. The Coordinate best representing a point feature (or a complex set of coincident point features).

b. The start or end coordinate of a segment.

c. The common coordinate (which may be selected from the Redundant Coordinate Set) at the point of intersection of two or more Segments. This may result in the subdivision of one or more of the intersecting segments.

<Edge> ::= <Redundant Coordinate Set><Segment Subset>

OR ::= [that part of a Segment which is not a Node. This term is equivalent to LINK in several sources]

<Chain> ::= <Node><Edge><Node><Edge Set>

<Edge Set> ::= <Edge><Node><Edge Set> OR <null>

<Polygon> ::= <Chain>

OR ::= [a minimal two dimensional surface which is not further subdivided by any Chain combination and which is bounded by a Chain]

For the remainder of this design document, these definitions will

be in effect.

2.1.4 Geometrical Attributes

A feature has been described as an entity located in space. Various constructs which might be used to describe the position of that feature have been considered and it has been implied that certain positional data might be considered as attributes. It is now necessary to consider how and where these geometrical attributes might be held.

Elevation. There are a number of ways in which elevation(s) could be held, and the following only considers a few of the options available.

a. Coordinate Attributes. The elevation(s) held could indicate one or more of the following z values:
   (1) Height of ground level above datum.
   (2) Height of base of Feature above datum and/or height above ground level.
   (3) Height of top of Feature above datum and/or height above base of Feature and/or height above ground level.

b. Node Attributes. As for coordinate attributes above.

c. Edge Attributes. As for coordinate attributes above, except that the values held will of necessity be of a statistical nature, eg. mean, maximum or whatever.

d. Area Feature Attributes. As for edge attributes above.

e. Line Feature Attributes. As for edge attributes above.

f. Point Feature Attributes. As for coordinate attributes above.

It should be noted that, unless the unit of measurement is fixed, some indication of the unit used (eg. feet, fathoms, meters) must be given within the attribute.

2.1.5 Attributes - General

Any point in the 'real world' has certain characteristics (which may be static or dynamic) associated with it. It is the recording of these characteristics with which we are concerned when considering attributes. So far, the term attribute has only been used with regard to those elements of positional data (such as Z) which are not to be considered as a part of the geometry of a map. The definition of attribute will now be extended to include all characteristics which are not explicitly geometric.

It should be noted at this point that the attributes which are to be captured for inclusion in the database will have been derived from a number of different sources, such as photo interpretation. Thus, they represent a number of selective perceptions of the 'real world'. It must be appreciated that this initial perception of reality should not be considered as a fixed perception. As new requirements (or data sources) are identified, then the database attributes specification will also require modification. The handling of attributes within the database structure must therefore be made as flexible as possible in order to minimize the impact of such modification.

Attributes may be objective or subjective, and they may be qualitative or quantitativ. However, they will always be described by a value (in some form) and a means whereby that value may be interpreted.

Attributes can be divided into at least three broad categories:

a. Physical. These attributes are those characteristics which, subject to generalization and an acknowledgement of

the sensing device being simulated, are directly observable 'on the ground' eg. Woods, Roads.

b. Calculated. These attributes are derived and as such are not directly observable on the ground. However, given the same input data the calculations should be repeatable. Examples are isolines, and population density.

c. Social/Political. These attributes are arbitrary in that they are defined for human convenience. They may or may not be directly observable, and they may be manipulated without affecting any 'physical' features, eg. County boundaries, Town names, Flight Information Regions.

2.2 Logical Schema for Data Structure

Figure 2.1 shows a logical schema diagram for the proposed data structure. Each of the circles represent a file (or a logical unit in a single file). These files could also be relations in a relational DBMS. In all the files, there are only three physical points which could be switched to logical keys in a relational environment (The coordinate start/stop and attribute pointers are the only physical record pointer). Each file and its interface, or relation to the others is now described.

2.2.1 Node File

The Node file contains information on topological nodes as well as artifact nodes created due to the requirements of the data structure. There are four node types:

- A normal topological junction where three or more edges intersect except in the case of a single edge feature.
- An artifact node generated by the software every 64 coordinate pairs for (x,y) data and every 32 coordinate tuples for (x,y,z) data. These artifacts are implemented to parse long coordinate strings into shorter logical

units that will speed certain areas of processing, such as overlay and proximity analysis*.

- An artifact node generated at the edge of a map sheet (or tile). These edge or boundary nodes, are used to tie sheets (or tiles) together as well as the features that cross sheet (or tile boundaries).
- An artifact node generated when an attribute changes along a lineal feature.

The node ID is a unique number ranging from 1 to n, where n is the largest positive integer a machine allows. This node ID is also a key. Thus, each key is unique.

2.2.1.2 Relationship to Other Files

The node file relates to three other files:

- _The coordinate file._ The actual geographic location of the node is stored in the coordinate file. While this is not optimal for data entry, it does logically separate topology from geography. For data entry, a special index file for node searches (storing coordinate data, ID, and physical record pointer) could be used for fast node searches. This file is described further on. Please note that this index file is not needed in the static user data base. Also note that the coordinate file relates to the node file (see coordinate file discussion).

---

* These number are for 32 bit representations. In using a 'local' coordinate system, 128 (x,y) pairs maximum would be stored per edge.

- The edge file. The node file relates to the edge file in terms of a list of edge ID's that are related to that node (regardless of node type). The edge file relates back to the node file.

- The feature file. The node file relates to the feature file in the case in which a node is also a point feature. In this case, the point feature is defined by the position of the node. However, the node is only related to a feature in that a node can exist as a topological entity with no spatial context.

Figure 2.3.2 describes the physical layout for the node file.

2.2.2 Edge File

The edge file contains references to all edges stored in the map sheet. This file is purely topological. As such, it is related to a number of other logical files.

- Node file: The edge file relates to the node file in two ways. The first is that each edge starts and stops at a node. Thus, the edge records contain the start and stop node ID's. By supplying this information, such analytical requirements as network analysis are facilitated. The second way in which the files are related is that each node can be associated with many edges. See section 2.2.1 for this discussion.

- Coordinate file: The edge file contains no coordinate information. This information is stored in the coordinate file. The edge file stores pointers into the coordinate file. Access to the coordinate data is through the edge file or the node file.

- Feature chain file: The edge file is not directly related to the feature chain file. However, the feature chain file contains a list of EDGE ID's for edges defining a feature.

- Feature file: The direct reference to an edge is through the feature file - feature chain file route. There is also a direct relationship between the feature file and the edge file through storing Left, Center, and Right feature ID's along with the edge record. Thus, each edge contains the knowledge as to which features it is associated with. This is important for spatial queries and reclassify/merge/dissolve operations.

Each edge is assigned a unique edge ID, starting at 1. The system maximum in a 32 bit environment is then 2.14 billion edges per theme. Each edge has a type code. This type code is related to precedence. In certain cases, such as plotting interstate overpasses, part of one road (or part) is to be masked by another road. Presedence codes may be used to enforce this condition.

The physical layout for an edge record is in section 2.3.3.

2.2.3 Coordinate File

The coordinate file stores all positional (locational) information. These are the actual (x,y,z) tuples that define the geographic location of nodes and the actual segment definition for edges. The node and edge files point into the coordinate file either through a key (relational implementation) or physical pointer (non-relational environment).

The coordinate file is related to 2 other files:

- Node file: The Node file points to the coordinate file. The Coordinate file points back to the node file through

the Node ID.

- Edge file: The edge file points into the coordinate file through start/stop pointers. The coordinate file points back to the edge file through the Edge ID.

Besides coordinate data, the coordinate file also stores a useage code and a entity key. For each logical entity defined by a coordinate, or coordinates, a header of useage/key is stored. Useage may be either Node or Edge. The key is the Node ID or Edge ID as indicated by the useage code.

A detailed discussion of coordinate storage may be found in section 2.3.4.

2.2.4 Feature File

The feature file is the locus for storing the spatial structure in the database. It is also the cross roads between the topologic structure and the attributes/cartographic database. The feature file is created by the verification process and any subsequent attribute tagging processes. As such, each feature record is comprised almost entirely of pointer keys into other files. These files are:

- Node file. The feature file points to the node file in the case where a node is also a point feature. In this case, the node file is also related to the feature file.

- Feature Chain File: The feature file contains a logical pointer into the feature chain file. The chain file (section 2.2.5) contains a list of edge ID's for those edges that define the feature (or regional complex). The Chain file points back to the Feature file. In this way one can enter the Chain file and determine what feature is associated with a chain of edges.

- Attribute file: The Feature file points into the attributes database through the use of a search key. This key relates the internal attribute key code to an external key, which could be an alphanumeric string, such as mapname and parcel number.

- Cartographic Symbolization file: There is a logical pointer that links a feature to its cartographic symbolization information.

- Label file: There is a logical pointer that relates a feature to its label annotation file. This file contains ancillary text, tables, or lead line information related to a feature.

Each feature is assigned a unique ID as it is added to the database. Each feature also has a data type - point, lineal, or area. Finally, the area/length of the feature is stored along with the feature. This is purely to avoid having to store this information in an external attributes database.

A detailed discussion of the physical layout of the feature file is in section 2.3.5.

2.2.5 Feature Chain File

The Feature chain file contains the knowledge of which edges are 'chained' together to form features. This file is created by the topologic validation/feature formation (verification) processor. The Chain file interfaces with two other files.

- The Edge file. The Chain file points into the Edge file. The pointers are an ordered, connected list of edge ID's defining a feature. The Edge file defines the characteristic of each named edge.

- The Feature file. The Feature file points into the Chain file through an ID that indicates the beginning of a chain list for a particular feature. The Chain file points back into the Feature file through the use of the Feature ID.

The primary function of the Chain file is to allow application software access to feature definitions.

Detailed layout of the chain file is in section 2.3.5.

2.2.6 Label File

The definition of the Label file is beyond the scope of this effort. However, for general discussion, the label file stores information related to a feature such as abstracts, numerical tables, special annotation and so forth.

2.2.7 Cartographic File

The definition of the Cartographic file is beyond the scope of this effort. This file contains information on the cartographic symbolization information and is related to a feature. Symbolization includes color, font, pattern, line weight, and orientation information.

2.2.8 Attribute File

The Attribute file contains all the numeric and character string information related to a feature. The attribute data defines the non-spatial/topologic characteristics of a feature. The Attribute file does not have to exist unless there is actual attribute data to store. The attribute information could be stored in either a relational DBMS or in a MOSS multiple attribute file format. The Attribute file's primary connection

is with the feature file:

- The feature file stores an attribute ID which is used as a logical pointer into the attributes file. The logical pointer can be either the Feature ID or a special alphanumeric key, such as map name and parcel identifier. The attribute file points back into the feature file by storing the Feature ID along with the row(s) of attribute data associated with that feature.

Section 2.3.7 describes the physical layout for a non-RDBMS implementation.

2.2.9 Other Files

There are three files not shown in the logical schema layout for the data structure. These three files are what ties the logical pointers (ID's) to physical addresses in an RDBMS implementation. The logical pointers become the indices between the different files (tables). In a non-RDBMS implementation, these files would become what we term as cross-reference files. There is one each for the Node file, the Edge file, and the Feature file. Any access based on a logical pointer would index into the appropriate cross-reference file to retrieve the actual physical location of the desired node, edge, or feature.

All three cross reference files have the same structure and are described in detail in section 2.3.10.

2.3 Physical Data Structure

This sub-section describes the proposed physical data structure for an arc/node MOSS.

2.3.1 File Organization

At this time, each map sheet is an independent file in the MOSS database. At most, each map has two files associated with it; the spatial/cartographic data and the attribute data. Each map has a name and the file is referenced using this name. Under the new structure, we propose that the map name still be used as the basic reference for storage and retrieval. This means that each map sheet will be stored on disk as an independent set of files. While this does not preclude all map sheets from being merged in a single file, it does allow for ease of implementation and continuity with the current mode of operation in MOSS. At some time in the future (2 to 3 years) the idea of storing all map files in one single massive database should probably be addressed. However, we believe that the maps should be stored as single entities. An indexing system could be implemented that ties related topological and cartographic entities between sheets. This is exactly what the map names file currently used in MOSS does. We suggest that the current map names file be used as is during the initial enhancement period (arc/node structure and relational DBMS interface) and that the effort be concentrated on the actual arc/node structure for individual map sheets (or multiple map sheets).

Under the new structure, each map will have up to nine files associated with it. These map files are:

- Map Header: This is a small file that contains information about the map: when it was created, projection, and so on.
- Edge file: This file will have information about individual edges, such as number of coordinate pairs, and logical pointers into the other files.
- Node file: This file will have information about nodes, such as their coordinate key and number of arcs associated with that node.
- Coordinate file.

- Feature file: This file has information about the spatial structure inherent in the topological structure. Such characteristics as # of arcs associated with a feature, feature type and logical keys to other files are stored.
- Feature chain file: This file will store information associated with the edges that comprise a feature.
- Index file: This file is an index file to the features in a map sheet. It is designed to be a pre-processor file to a map file before any coordinate data is actually processed.
- Symbolization file: This file has information on cartographic symbolization, such as line weight, line style, and line color.
- Attribute file: This file has information about what attributes are associated with a map. These attributes could be feature area, forest stand type, road surface type, or whatever other attributes are associated with a feature on the ground. This file could be easily stored in a relational DBMS.

At the very minimum the header, edge, node, coordinate, and feature files are required for a map.

2.3.2 Node File

The node file contains information on what edges meet at a common point. This file is necessary to fulfill the requirements of topology while also allowing for tying edges together to form features.

The node records are each 64 bytes. We suggest storing 8 nodes per physical record on disk. Thus each physical record would be 512 bytes.

The node record can have feature associations. As recognized by such groups as USGS (DLG format), nodes, while dimensionless, can

have attributes. If the node is also a point feature, the feature ID is the reference into the feature file. If the node belongs to more than one feature, then the feature ID becomes a pointer into the feature chain file for a list of feature ID's that the node is related to. The number of edge ID's is the number of edges associated with that node. The coordinate ID is the reference into the coordinate file where the node coordinate is stored.

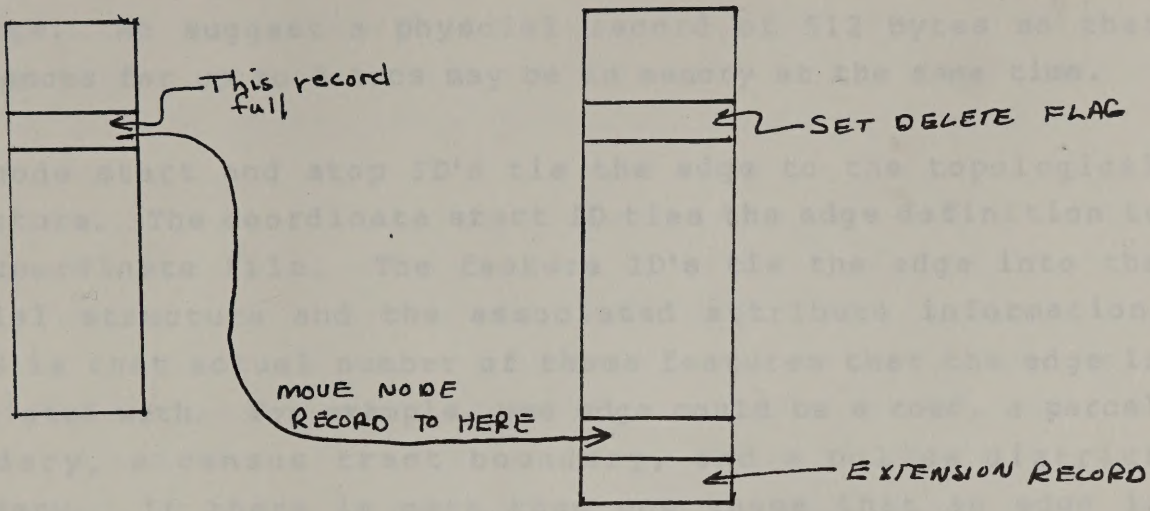
Under this structure, it appears as if each node has a maximum of ten edges. While this would satisfy any and all natural resource applicatons, there are cases where more edges could be associated with a node (a traffic circle).

When the software attempts to add an eleventh edge to a node, then the following sequence would occur:

1. The node record would be copied to the next available location at the end of the node file.
2. The node counter would be updated.
3. The delete flag would be set for the old node record.
4. The edge pointers would be used to read records from the edge file so that the node record pointers can be updated to reflect the new node record location.
5. The tenth edge record would be moved into the record immediately following the new record location for the old node.
6. The cross reference file is updated.

This sequence would provide space for another ten arc pointers.

Figure 2.2 shows the organization of the node file.


This record full
SET DELETE FLAG
MOVE NODE RECORD TO HERE
EXTENSION RECORD

The extension record would have the same record layout as an ordinary node record except that the NODEID is zero.

If a node is deleted from the database,then the NODEID is set to the maximum negative value allowed by the system.

2.3.3 Edge File

Figure 2.3 has the layout for the edge file.

Each Edge record contains information related to that edge and it's relationship to other files. Each edge requires 64 bytes of storage. We suggest a physcial record of 512 bytes so that references for up to 8 arcs may be in memory at the same time.

The node start and stop ID's tie the edge to the topological structure. The coordinate start ID ties the edge definition to the coordinate file. The feature ID's tie the edge into the spatial structure and the associated attribute information. NFIDS is that actual number of theme features that the edge is associated with. For example, one edge could be a road, a parcel boundary, a census tract boundary, and a police district boundary. If there is more than one theme that an edge is associated with, then the related feature ID's are stored in the feature chain file and not in the edge file. Please note that an edge can have left, center, and right features associated with it. The MBR of the edge is the minimum enclosing rectangle. This information is used for improving spatial processing. The X and Y bands are described in section 2.3.5.

If an edge is deleted from the database, then the edge ID is set to the maximum negative value allowed by the system.

2.3.4 Coordinate File

The coordinate file contains the (x,y) or (x,y,z) coordinate pairs associated with the edge and node files. Chrisman (1984) provides a thorough discussion on storage techniques for coordinate data in a geoprocessing environment. It is interesting to note that MOSS has traditionally used a hybrid storage system - local coordinate offset from a feature MBR. This is also the technique favored by others for coordinate compaction (Chrisman, Niedzwiadek). The concept of a local coordinate system by feature is fine for resource applications.

Chrisman, N. 1984 Title, Reference

However, for other applications, such as those related to surveying, it is totally unacceptable. Therefore, we propose that MOSS allow 2 forms of coordinate storage. One would be full 32 bit integer storage and the other would be local offset storage based on sheet center instead of the feature origin.

In the 32 bit storage environment, any coordinate location on the surface of the earth can be modeled to an accuracy of ~.36 of an inch. The drawback to this stoarge technique is the space required to store the coordinate data. The advantages are that it is simple, consistent, and easy to understand. Also, since no transformations are required to go from local to global coordinates, it is compute efficient. The decimal place location is specified in the map header. If greater precision is required, then the position of the decimal is important. The default is no decimal place.

In the local coordinate storage technique, the sheet center coordinate values are used as a local origin. Each feature coordinate is then subrtracted from this and multiplied by a scale factor to achieve a 16 bit X and a 16 bit Y value for storage. This is the technique used in AMS now. The primary advantage is that the size of the coordinate file is halved. This can be significant in large files (for 100,000 coordinate pairs, we will see a savings of 400,000 bytes of storage). The disadvantages of this storage technique is the requirement for math computations, which will make it more computation bound.

The coordinate files will be a random access file where each logical record is an (x,y) or an (x,y,z) coordinate. Each coordinate string will have a 16 byte header. This header contains several pieces of information. The first is a coordinate set ID. This ID uniquely identifies that coordinate set. It starts with 1. The second piece of header information is the useage. Is this coordinate set for a node, edge, or label location. The third field is the storage node. Some examples

are given. Others could be added.

There is no indexing other than the node or edge key in the coordinate file. Indexing is maintained in the edge and node files. When an edge or node is deleted, the coordinate data will stay in the coordinate file but will not be referenced.

Each physical record for coordinate data will be set at the system physical disk record size.

2.3.5 Feature File

The feature file maintains information on each feature in a map sheet. It does not contain information on what edges are associated with a feature. These are stored in the Feature chain file.

The layout for the feature file is in Figure 2.5.

Each logical feature record is 64 bytes. Each physical record for storage is the physical record size (usually 512), allowing for 8 feature records to be in memory at any one time.

Most of the fields specified for a feature record are straight forward. However, several of them require further clarification. First, the pointer into the cartographic symbolization file is used only if there is a cartographic file. This file is not required and is in fact beyond the scope of this effort. As such is will not be mentioned from here on in. Suffice to say that the cartographic file contains information on line style, color, hatching, label locations, and so on.

The fields labeled XBAND and YBAND are the same as those in the edge file. Each map sheet is divided into 32 horizontal bands

and 32 vertical bands. When a map is added to a MOSS data base, each feature (or arc) will be tested to determine which bands the feature (or arc) fall in. This is done with very fast MBR checks. The bands are numbered from 1 to 32 (North to South or East to West). Bits are set in the band word according to which band(s) the feature (or edge) falls in. The band information is used to pre-process the data to limit candidates before any actual display or analysis begins. For example, in the overlay process the band specifications for a feature in map A can be compared with that for a feature in map B using a Boolean AND. If the result is 0, then proceed to the next comparison. If the result is non zero, perform an actual MBR check. Banding is simply a technique for speeding processing efficiency.

The minimum and maximum X,Y, and Z values are the minimun bounding rectangle for the feature. The MBR is used to increase processing efficiency.

The Attribute ID is a reference into the attributes database. For the sake of simplicity and ease of algorithmic implementation, there is a one to one correspondence between the feature file and the attributes file. For example, if there are 200 features in the map, then there will be 200 attribute records. Thus, area, length, perimeter and so forth could be stored in the attributes file.

Lastly, the area and length data are stored in this file so that the majority of the MOSS analysis and display functions can be used without the need for an attribute file.

2.3.6 Feature Chain File

The feature chain file maintains information on the edges associated with each feature in a map sheet.

The layout for the feature pointer file is shown in Figure 2.6.

Islands are indicated using a value of 0 to separate the list of edge ID values. The outer feature boundary for area data is called a parent. The parent edge sequencce is terminated with a 0. Island pointers follow. The FID relates back to the feature that a set of edges belong to.

Obviously, the number of edges per feature can vary. To save space, the file can be thought of as a series of 4 byte logical records. The physical record length will be the largest physical record unit on the disk (usually 512). In this way, random access to fixed length records can be used even though the logical feature pointer records are variable length. Thus, special file access and processing can be avoided.

2.3.7 Attribute File

The attributes file is used to store attribute information for a feature and its associated edges. This file is not required (some maps have no attributes or only a graphics representation may be required). However, if there is attribute data, then there are two possible ways in which they will be stored in MOSS. The first is to store the attributes data in the existing MOSS attribute structure.

Under the current MOSS structure, up to 200 attributes per feature may be stored. Appendix A has a list of functional capabilities and commands for attributes data currently supported by MOSS. This attributes file may be thought of as a single relation table in a relational DBMS. It is organized in a row-column format with each row corresponding to a feature and each column corresponding to a variable. Each map has its own attributes file. Appendix B has the current file structure.

The second method is to store the attributes in an external DBMS. To date, this has been accomplished on a rudimentary basis using a file linkage between MOSS and INFOCEN. 3CI has a publication that describes this linkage in detail.

Interaction with the attributes database occurs at two levels: low frequency interaction and high frequency interaction. Low freqency interaction happens when the user points to a feature with the cross hairs and asks for that features attribute(s) to be printed out. The current 3CI link between MOSS and INFOCEN is a low frequency interaction.

High frequency interaction occurs when many attribute records must be accessed and manipulated in a short period of time. This type of interaction would occur in a function such as overlay. Obviously, in a high frequency environment the use of intermediate disk files for pointer transferral is not feasible.

A much closer tie is being required between MOSS and the DBMS. The question becomes 'How close?'.

The key design consideration for the interface from MOSS to the DBMS is that of maintaining a generic interface. We do not want to tie MOSS explicitly to only one DBMS, this would never satisfy the requirements of the user community. Additionally, 'OSS would then be tied to only one system with no room for growth as well as being totally dependent on the vendor of that DBMS. Finally, many groups using MOSS cannot afford to purchase proprietary software.

What this tells us is that the interface can be direct but through a DBMS shell using only the basic DBMS operations required by MOSS. These basic DBMS operations would be:

1. open the attributes database,
2. validate field name,
3. read a row based on a key(s),
4. write a row based on a key(s),
5. get field value from row,
6. store field value in row,
7. delete a record based on a key(s),
8. search for occurence,
9. logical schema manipulation,
10. close the database.

The DBMS, from MOSS's point of view, becomes a primitive depository of attribute data. The higher level attribute manipulation functions would reside in MOSS, not the DBMS. Such things as boolean evaluation, area reports, report generation, and so on would remain in MOSS and act, from the users' point of view, just as they now do. There are a number of reasons for using a DBMS shell:

1. The current MOSS routines can easily be changed to use the same parameter list as those required by the

interface to the DBMS, allowing the users greater flexibility.

2. The overhead of keeping a complete DBMS under MOSS is minimized, thus allowing MOSS to run on smaller, less expensive computers.
3. The maintenance of the shell will require less effort.
4. MOSS is much more protected from the 'whims of change' when a vendor decides to make some radical changes to the DBMS.

The DBMS shell is described in section 4 (subroutine descriptions), and is based on evaluation of the RIM and INGRES FORTRAN interface routines.

Please note that the user can always execute the actual DBMS to use the more powerful capabilities (if required).

The external database is accessed by passing a key. This key is a combination of the map name and unique feature number (or name).

Figure 2.7 has the layout for the non-RDBMS implementation. Access would still be through the DBMS shell so that MOSS itself is independent of the attribute source.

The number of attributes tells the software where to begin reading the actual attribute data. Field name is a 1 to 16 character key that is used for searches. Field type tells the software if the data are whole numbers, floating point numbers, character string, or time/date. The maximum number of bytes in one row of attribute data is 8192 bytes. The actual record length for a logical record is stored in the attribute header. The unique identifying ID is not a direct link back to the feature file. Instead, it is a logical pointer. Many feature can be related to a single attribute row through the use of these logical pointers.

2.3.8 Cross Reference Files

Figure 2.8 depicts the layout of the node, edge, and feature cross-reference files. These files are used to link logical pointers to physical records. The path sequence is always to extract the logical key from the appropriate file, use it as a direct index in the cross-reference file, extract the physical record pointer and finally read the pertinent record from the desired data file.

2.3.9 Fast Index File

In order to rapidly process the feature data without actually going to the feature file, an index, file can be used. Whenever spatial queries or feature comparisons (such as in overlay) are required, then this file is scanned before actually accessing the coordinate database. This file would look much like the current MOSS fast file, allowing for compatibility with existing databases and software. In the current MOSS environment, the 'fast' file allows the software to scan between 1000 and 2000 features per wall clock second (depending on the CPU). The structure for this file is shown in figure 2.9.

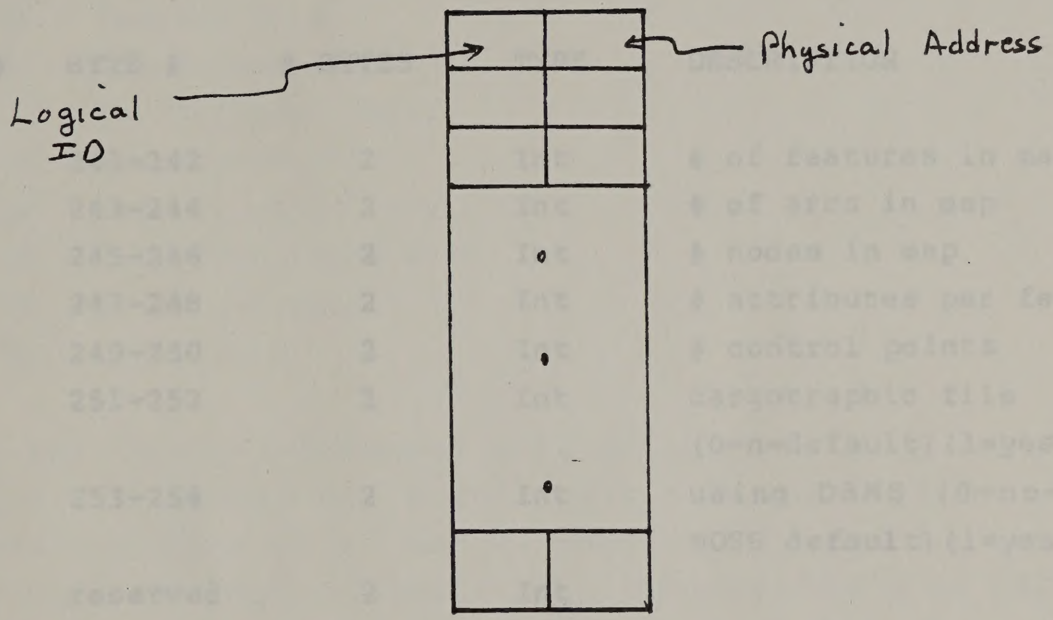
This file is created when the map is added to the database. This is not a fancy indexing scheme into the database but is fast, simple, low overhead and reliable.

As a point of reference, we have studied the use of other indexing techniques. From the literature we reviewed, while many prototype systems using these techniques exist, none are production systems for vector data. Additionally, none have been tested on large (more than 1000 features) datasets. While these techniques definitely warrant research, at this time their operational characteristics are still an unknown for vector data and as such should not be made part of Phase I upgrade for MOSS.

Figure 2.8

Cross Reference File Layout

All three reference files have the same layout. Each logical record is 8 bytes.



ID is the node, edge, or feature ID, which are integer numbers from 1 to n, where n is the maximum positive integer allowed by the compiler. The ID's are stored in ascending sorted order. The physical pointer is the physical record number of the desired node, edge, or feature.

2.3.10 Map Header File

The arc/node map header will be exactly the same as the current map header in MOSS. Thus, the map header will be a minimum of two records, each 256 bytes in length. Additional records may be used for general descriptive information about the map.

Figure 2.10 is the layout for the current MOSS header. Below are the additions to the header for the arc node structure.

WORD #	BYTE #	# BYTES	TYPE	DESCRIPTION
121	241-242	2	Int	# of features in map
122	243-244	2	Int	# of arcs in map
123	245-246	2	Int	# nodes in map
124	247-248	2	Int	# attributes per feature
125	249-250	2	Int	# control points
126	251-252	2	Int	cargotraphic file (0=n=default) (1=yes)
127	253-254	2	Int	using DBMS (0=no- use MOSS default) (1=yes)
128	reserved	2	Int	
129-152	257-304	48	Int	up to 12 control points
92	183-184	2	Int	Coordinate storage type (0=32 bit int) (1=local)
93	185-186	2	Int	Decimal place position (0=none, 1=one, 10=tems)

The reasons for not modifying the current headers are:

1. All current MOSS MAPS (numbering over 50,000 now) are usable.
2. Minimum changes to the current software on the D.G., DEC, and HP environments.

2.3.11 Map Name Conventions:

In order to uniquely identify each of these files on disk, the following disk file names will be used:

- FH ~~HD~~ - Map header. This extension is already used in MOSS to signify a map header.
- FE ~~ED~~ - Edge file.
- FN ~~ND~~ - Node file.
- FF ~~FT~~ - Feature file.
- FC ~~CO~~ - Coordinate file.
- FG ~~CT~~ - Cartographic file.
- FA ~~AT~~ - Attributes file.
- FL ~~PT~~ - Feature chain file.
- SF ~~FS~~ - Fast file/index file.
- FR ~~CR~~ - Cross reference
- FT ~~LF~~ - Label file

These two letter extensions will be added to a map name as an extension to create the actual disk file name. For example, suppose the name of a map is WHERE. The name of the file containing the edge information for this map would be WHERE.ED. This may seem like a large number of files to have for each map. However, each file is a distinct logical unit and could be thought of as a relation in a relational DBMS. Also, for any given function in DELTA ~~MOSS~~, these files will never have to be open all at the same time. Subroutine SUFF is used to add the extensions to a map name. A sorted list of map names is stored in the Project .PN file

2.4 Summary

This section presented the logical and physical layouts of the files required for an arc/node implementation in MOSS. The interfaces between the various files was graphically depicted. The next section describes in detail how the data structure is related to the processing modules described in section 1.6. Structure diagrams, interfaces, and algorithms are given.

3.0 Logical Processing Modules - Detailed Discussion

This section details the following logical processing modules.

- Attribute Manipulation,
- Data entry,
- Verification/Polygon Formation
- Overlay
- Zone Generation
- Polygon to edge
- Vector to raster
- Edge Intersection
- DBMS Shell
- Database primitives

Details include: flow diagrams, algorithms, data structure interfaces, and hooks into the current MOSS.

Structure diagrams are used to logically depict data flow, processing sequence, and interface points into the database. Certain symbols are used to graphically portray certain information. Rectangles are local processing units. They could be thought of subroutines. Ellipses are used to show interaction with the database. Lines with arrows show information flow. The numbers associated with the arrows relate to a table that describes information packets. This is known as interface information. The circled numbers in the rectangles show program flow. All diagrams start on the right hand side at the 3 o'clock position and are read in a clockwise order.

3.1 Attribute Manipulation

3.1.1 Attribute Reclassification

3.1.1.1 Description

Attribute reclassification is the procedure of changing values in fields of attribute data based on some set of rules. For example, the rules might be: if the current values are from 1 to 10 change them to a 1, if they are between 11 and 20, change them to a 2 and so on. Reclassification can either change values in a current field or create a new field based on existing values. Attribute reclassification is an integral part of the reclassify/merge/dissolve operation. Figure 3.1 is a structure diagram for attribute reclassification. Figure 3.2 is suggested user interaction.

3.1.1.2 Reclassification Interface Information

1 = Logical unit for input
2 = Field name to reclassify
3 = Flag for change current field or create new
4 = If new name of new field
5 = If new description for new field
6 = Reclassification rules
7 = FID number
8 = Field value
9 = Name for field
10 = Error return
11 = New field value

3.1.1.3 Reclassification Flow Sequence

1. User input for reclassification rules
1.1 Validate user input - check against database
2.0 Read an attribute-Go to feature file of desired map sheet to get the attribute key. Then go to the attribute file to get the attribute value for

the specified key

3.0 Apply reclassification rules to value

4.0 Insert new field value back into database. Either use update function or insert in new field

5.0 End

3.1.1.4 Attribute Reclassification

1 - Enter field name
2 - Do you wish to change existing field or create a new one?
3 - (If no to 2, skip to 7)
4 - Please enter name of new field
5 - Please enter description for new field
6 - Please enter data type
7 - Please enter reclassification rules in either the form new = old to old or if-then-else constructs.
8 - Please review input. Do you wish any changes?
9 - Enter layer and map sheet name to apply reclassification to

3.1.2 Attribute Entry/Validation

3.1.2.1 Description

Attribute entry/validation is the process where by the user enters attribute information into the database management system. If attribute entry is through an external RDBMS, then the reader should review the pertinent vender supplied manuals. This document will describe only attribute entry related to an internal implementation of a DBMS in MOSS. Since the existing internal DBMS is considered adequate, only enhancements and reconfiguration are deemed necessary. A whole new implementation is not necessary. Figure 3.2 shows the logical structure of the attribute entry for MOSS.

Module A allows the user to build a definition of the attribute tables to be entered into the database. This definition includes data type, allowed range, key, and description. This software already exists in MOSS and will not be described further until the implementation discussion.

The Add module (B) controls the entry of the new attribute data into the database. It solicits all the required input from the user, validates file names, status, protection, and so forth. This module already exists. Control is then passed to the Sort/Merge (C) module.

The Sort/Merge (C) module uses the user input to control the actual add process. In this module, the attribute rows are matched against the feature file for exact match. Exact match is performed using the unique match key. This match key could be a number, map name and feature number, or any other unique key. This key is the first field in the attributes file. This same key is supplied for each feature at the time of map data entry.

3.1.2.2 Interface Information

1 = Parameter list from add module to be sent to Sort/Merge module.
2 = Error return.
3 = Channel numbers for definition file, feature file, attribute table, and final attributes file.
4 = Mismatch errors from Sort and Merge module.
5 = Data Channel for logical schema file.
6 = Error return for schema build module.

3.1.2.3 Software Flow

1. Build/edit logical schema definitions file
2. Solicit user control information in add module
3. Use user supplied information to open required files, check

security flags, and validate that enough space exists in database.

4. The Sort/Merge module performs the following:

   a. get attribute key from feature record
   b. use cross reference
   c. scan attribute table for match
   d. when match, read row from table
   e. validate and parse row based on schema definition
   f. insert row into database
5. End

3.1.2.4 User Interface

User interface for schema file.

1. Enter name of schema file?
2. Add, Edit, Help or Bye?

   If Add
3. How many fields are to be defined?
4. Enter field name.
5. Enter field description.
6. Enter type (Int, Real, Char, Time/Date)?
7. Enter number of characters in field?
8. Valid lower value (None for NONE).
9. Valid upper value (None for NONE).
10. Enter field names of other fields this field related to (None for NONE).

    If Edit
1. Enter field name to edit?

   Jump to attribute add sequence.

User Interface for Add module.

1. Enter name of schema file.
2. Enter name of attributes input file

3. Enter name of map that attributes are to be associated with.

3.1.3 Attribute Update

3.1.3.1 Description

There are two types of attribute update. The first is to update the contents of a field for all features in a map. The second is to update a single value in a single field for a single feature. The first type is already supported in MOSS through the attribute program. The sequence described in 3.1.2 is valid for update. Single feature update is already supported in MOSS through the QUERY command and as such requires no further discussion.

See Implementation (Section 4) for integration/implementation notes.

3.1.4 Delete

3.1.4.1 Delete Description

The attribute delete module allows the user to 1.) delete an entire attributes file or 2.) delete one field from an attributes file. Deletion for a feature is not necessary: by definition, when a feature reference is deleted, the attribute information, while still in the database, is no longer referenced.

Deletion of the entire attributes file is already implemented in MOSS and is not further described.

Deletion of a field in the attributes file is shown in Figure 3.4. The user is first asked for relevent information required for the deletion. These data are passed to the Delete module

which zero's the specified field. Next, the attributes file is compressed to free the space left from the delete process. Finally, the attribute header is updated to reflect the delete.

3.1.4.2 Delete Interface Information

1. Field name
2. Field type
3. Logical unit for attributes file
4. Error returns
5. Field location

3.1.4.3 Delete Software Flow

1. Get field name and validate that is exists
2. Get field type and length if key is valid
3. Use field key to tell delete module which field and its location so delete can take place
4. Once field is deleted, compress the attributes file to free space. Compression occurs as follows.
    a. start at begin location for deleted field
    b. move all fields after deleted field up 1 location (if field 2 is deleted, move 3 into 2, 4 into 3, and so on)
    c. temporary file may have to be used to ensure that the operating system knows that space is being saved
5. Update attribute header as follows:
    a. delete reference to deleted field
    b. change field location references for all fields stored after deleted field.

3.1.4.4 Delete User Interface

1. Please enter map name
2. Please enter name of field to delete
3. Are you sure you want to delete?

3.1.5 Merge

There are three types of attribute merge. The first type occurs in an operation such as the intersection option in overlay. Under this scenario, a row of attributes from map A is merged horizontally with a row of attributes from map B. The second type would occur in a map merge operation. This is horizontal map merge. Under this scenario, the attributes for map B are appended on a row by row basis at the bottom of the attribute file for map A. The third type occurs in the union operation in overlay. In this case, a feature takes on the attribute rows of its composite features. Thus, a number of rows would have the same feature ID, would be located contiguously in the file but could have different lengths. Figure 3.5 pictorially shows the three attribute merge types.

3.1.5.1 Merge Description: Type I

Type 1 attribute merge allows the user to merge attributes for two maps into a single row of attribute information. This merge process will occur on a regular basis in a function such as polygon overlay using intersection or map merge. In this operation, if map A has 5 attributes per map feature and map B has 7 attributes per feature, then the new map could have as many as 12 new attributes per feature. However, the user may wish to have no attributes passed along, the attributes for only map A, the attributes for only map B, or a combination of both. Therefore, it is necessary to give the user flexibility in defining which attributes should be passed through the merge

processor. (see 3.1.5.4).

Please note that Type 1 attribute merge already exists in MOSS. However, due to its importance in the overlay processor, merge is described further here. See Figure 3.5.1

3.1.5.2 Interface Information

1 = Map name for map A
2 = Map name for map B
3 = Logical unit number for attributes file
4 = Returned list of fields to merge for map A
5 = Returned list of fields to merge for map B
6 = Flag = 0 save none, 1 = save from A, 2 = save from B, 3 = save both A and B
7 = Error flag
8 = New map name
9 = List of feature keys map A
10 = List of feature keys map B
11 = Retrieval/process criteria

3.1.5.3 Merge Software Flow

1. Open attributes files for map A, B.
2. Get list of field names from user from map A and map B
3. Validate field names
4. Create new attribute file entry in attributes database and initialize
5. Create a list of feature keys for map A and/or map B based on retrieval/process. C is a generic module and can refer to overlay, proximity, edge, or simple map name. Please note that the "merge" attribute function operates the same in either a one map or two map situation.
6. Add attributes information to the new attributes dataset

in the attributes database.

7. Update attributes header to reflect the data added into the new map.

3.1.5.4 Merge User Interface

1 Retain attributes map A? (CR means all)
2 Retain attributes map B? (CR means all)
3 Enter field names for map A
4 Enter field names for map B

3.1.5.5 Attribute Merge: Type II

Type II attribute merge occurs when two maps are merged into one map file. In this case, each map must not only have the same number of attributes but also the same attribute fields in the same order. In this case, the attributes for Map B are appended to the end of the attribute set for Map A. Figure 3.5.2 has the structure diagram.

3.1.5.6 Interface Information

1. Data channel map A and B.
2. Work space.
3. Error return = # fields not same.
4. Field names from attribute files.
5. # fields map A + B.
6. # features in map A.
7. # features in map B.
8. # bytes per attribute record.
9. Attribute read/write error.
10. Data channel for.

3.1.5.7 Merge Software Flow

1. Open attributes files for Map A and Map B.
2. Check list of field names for Map A against those for Map B.
3. Create new attributes file for Map C and initialize this file.
4. Get attribute rows from Map A and load into Map C.
5. Get attribute rows from Map B and load into Map C.

3.1.5.8 Attribute Merge: Type III

In this type of attribute merge, the new feature can take on all the attributes of the features it was composited from. This will occur in the polygon union process as well as the merge/dissolve process. The user interface, information flow, and information interface are exactly the same as for Type I attribute merge. The only difference is how the new map data are stored in the database. The easiest way is to simply store the input map name(s) and feature ID(s) of the original features along with the relation names (for an RDBMS implementation) rather than create relations with variable length records.

3.1.6 Attribute Dump to Interchange File.

3.1.6.1 Description

This module allows the user to dump attribute data from the internal MOSS storage format into an ASCII text file that can be used as input into some other DBMS.

3.1.6.2 Dump Interface Information

1 = Logical unit number
2 = Map name
3 = # Attributes
4 = Error return
5 = Attributes schema file name

3.1.6.3 Dump Software Flow

1. Get map name from user
2. Validate that map has multiple attributes
3. Retrieve # attributes from map header
4. Dump attributes to ASCII file using attribute logical schema rules in reverse
5. End

3.1.6.4 Dump User Interface

1. Please enter name of Logical schema file
2. Please enter name of map (table) to dump
3. Please enter name of file to dump attributes to

3.1.7 Attribute Browse

3.1.7.1 Browse Description

Attribute browsing allows the user to determine what attributes are associated with what features in the database. The browse function is currently operational in MOSS. It is not discussed further until implementation (section 4).

3.1.8 Attribute Report Generation

3.1.8.1 Report Description

Attribute reporting allows the user to create formatted tables of attribute information. Report generation is already supported by MOSS. Except for the user interface, report generation is not discussed further until implementation (section 4).

3.1.8.2 User Interface (current environment)

```
ENTER COMMAND
? REPORT
ENTER ACTIVE MAP I.D TO REPORT ? 1
ENTER NUMBER OF COLUMNS IN TABLE ? 3
DO YOU WISH POLYGON AREAS OUTPUT ? Y
DO YOU WISH SUBJECTS OUTPUT ? Y
FOR COLUMN 1
RETRIEVE ATTRIBUTE FIELD BY 1-ID 2-KEY ? 1
ENTER THE ATTRIBUTE ID ? 8
FOR COLUMN 2
RETRIEVE ATTRIBUTE FIELD BY 1-ID 2-KEY ? 2
ENTER THE ATTRIBUTE KEY ? HOMERANGE
FOR COLUMN 3
RETRIEVE ATTRIBUTE FIELD BY 1-ID 2-KEY ? 1
ENTER THE ATTRIBUTE ID ? 3
```

If the user desires more sophisticated report generation, then the report generator in an RDBMS should be used.

3.1.9 Attribute Retrieval

3.1.9.1 Attribute Retrieval Description

Retrieval based on attributes allows the user to use rules, such as boolean expressions, to retrieve information from the database. Retrieval is already supported by MOSS. It is not discussed further until section 4 (Implementation).

3.2 Data Entry

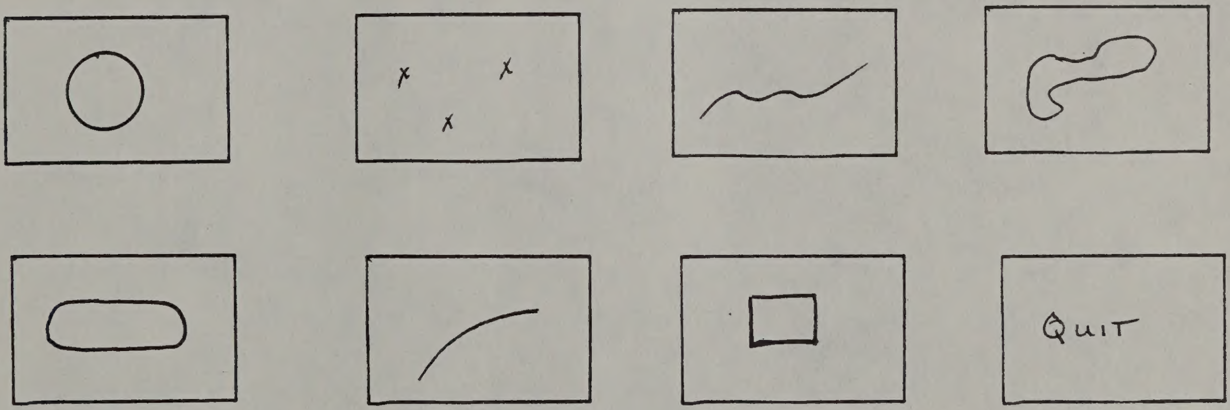
This section describes in greater detail how certain types of map data entry are facilitated under the new arc/node environment.

3.2.1 Geometric Shapes

Geometric shape generation refers to the ability to create rectangles, circles, ellipses, and so on at a terminal or at the digitizing table. Currently, MOSS supports this capability. Section 4, Implementation, describes how this function will be interfaced to the arc node structure. However, we feel that the user interface should be changed. This is described below.

3.2.1.4 User Interface:

A menu of geometric shapes is presented to the user. The user choses one shape using a pick device. The menu of shapes is:



Once the user picks a shape, the software prompts the user for the location and attribute information. These prompts are:

1. Rectangle
    - Point to Lower Left Corner
    - Point to Upper Right Corner
2. Circle
    - Point to center of circle or point to extreme radius
    - Enter circle radius
3. Line
    - Digitize points on line until done. Hit "end edge" button to terminate
4. Polygon
    - Digitize points on polygon boundary until done. Hit "end edge" button to terminate
5. Ellipse
    - Digitize focii
    - Enter length of minor axis
    - Enter length of major axis
    - Or point to major and minor axis limits
6. Points
    - Digitize points. Enter "end edge" button to terminate
7. Arc
    - Digitize center of curvature
    - Digitize end points of arc

These simple functions are the basics for a CAD type capability in MOSS.

3.2.2 Keyboard Entry of Geographic Data

3.2.2.1 Description

This capability already exists in MOSS. The question becomes how to interface it to the new arc/node and attribute data structures. This is described in section 4, Implementation.

3.2.3 Edgematch between Adjacent Sheets.

3.2.3.1 Description

Theoretically, edgematch should be performed during data entry. However, many systems do not support adequate edgematch. Therefore, MOSS requires some mechanism to match nodes/edges that cross map sheet boundaries. Figure 3.7, 3.8, and 3.9 shows the basic flow of this process.

It should be noted that this initial edgematch capability will perform attribute consistency checking across map sheets. This is easily done since we know the nodes on the boundary as well as the edges that is associated with the boundary node. Since the edge file contains Feature ID's, we have a path from the edge to the feature to the attribute file. Hence, attributes can be checked.

3.2.3.2 Edgematch Interface Information

1 = Channel number to map names file
2 = Returned valid map name to edge match
3 = Error return
4 = Logical unit numbers to open database files
5 = List of adjacent sheets before feature check
6 = MBR of sheet being inserted in the database
7 = List of candidates
8 = List of adjacent sheets
9 = Sorted list of adjacent sheets
10 = One adjacent sheet ID
11 = List of unmatched nodes

3.2.3.3 Edgematch Software Flow

1. Get name of map from user to perform edgematch on. (target map)

2. Validate name against map names file.
3. Get MBR of target map.
4. Determine sheets adjacent to target map.
    a. Open a candidate map
    b. Extract MBR for candidate map
    c. Check MBR's
    d. If check is true, add candidate map name to list of candidates
    e. When sheet compare complete, go to feature compare by sheet
    f. In feature compare, loop 1 to n through the candidate sheet fast file comparing each feature MBR to the target sheet MBR.
    g. Based on F, throw out spurious candidate sheets to build list of up to 4 valid adjacent sheets
    h. Make sure candidate sheet names sorted by East, South, West, North
5. Enter edgematch update processor. This processor performs the following:
    a. Open adjacent (candidate) sheet node file
    b. Scan candidate node file and compare against target map for the appropriate edge. A tolerance can be used so that an exact match is not required
    c. If nodes within tolerance, move target node to same exact geographic location as candidate node. Update coordinate record
    d. If nodes not within tolerance, add Node ID to list of no-matches
    e. At end of compare, if all match, do next sheet boundary. If some nodes don't match, continue
    f. Display unmatched nodes. Show candidate in one color or font. Show target in same color or font. Show distance separation
    g. Allow user to: 1) ignore and go to next, 2) move target to candidate using a pivot point, 3) move candidate to target using pivot point, 4) move both using pivot

    points.
    h. Update coordinate file with new node geographic position
6. Finally, perform horizontaal attribute consistency check.
    a. From Node information, get appropriate Edge IDs from Map A and Map B.
    b. From Edge information get Feature IDs from Map A and Map B.
    c. Through feature, get attribute records for Map A and Map B.
    d. Check attribute records for consistency (same check as in Type II attribute merge).
    e. If error, tell user. If not proceed to next boundary node.
7. Done

3.3 Verification/Polygon Formation

We propose that the current AMS verification procedure as it exists on the HP-9000 be used for the verification/polygon formation software module for the enhanced MOSS. Section 4 describes how the software will be upgraded for the new enhanced arc/node structure. Appendix C has a detailed description of how AMS verification operates.

The one major enhancement of the current AMS verifier is a check for spurious intersections of edges in the map. These spurious intersections can occur in one of two circumstances. The first is at a node. One edge can overshoot the node, thus intersecting another edge that also is associated with that node. The software should automatically correct this case without intervention by the operator. The other case occurs when two

edges intersect between node locations. This can happen due to sloppy digitizing. We propose to use the algorithm developed by USGS staff as part of the DLG3 data entry production system and as enhanced by EDC staff.

3.4 Arc/Node Overlay

3.4.1 Description

The overlay processor allows the user to overlay two separate map themes in the database to create a new integrated map. The overlay processor actually controls a number of other processors. These are: 1) user input, 2) a pre-processor to determine candidate edges, 3) an edge intersector, 4) building new map, 5) verification/polygon formation, and 6) build new attributes file.

3.4.1.1 Overlay User Input Description

The user must have total control over the overlay process. Within the current MOSS environment, selection of the features to be overlaid has already been accomplished before initiating the overlay process. The following functions may be used to pre-filter the data before overlay.

- Attribute selection (simple or boolean);
- Size and/or length
- Proximity

- Contiguity
- Geographic window

Using filtering functions allows the user to subset data into "active" data sets that will be more ammenable to the overlay process. Nevertheless, the user still needs to be able to specify:

- Maps to be overlaid
- Boolean operaters between maps
- Which, if any, attributes are to be carried through the overlay process.
- What the new map reference is to be called,

This is part of the user interface to the overlay process. Figure 3.10 shows the user interface processor configuration.

3.4.1.2 Interface Information - User processor for Overlay

1. Logical unit for map names file (or active table),
2. Expression for controling overlay process. See User Interface
3. Error codes
4. Stack of operands
5. Stack of map identifiers
6. # of map identifiers
7. List of attribute rules for merging attribute files after overlay (see attribute merge for more detailed discussion)
8. Array of attribute type codes
9. Array of attribute locations

3.4.1.3 Software Flow

1. Ask user to enter new "map" name.
2. Validate new map name against list of valid names (FNDMAP)
3. Get list of maps to overlay, including the boolean operations

to use with them.

4. Validate list of active map ID's or map names to make sure they exist. If they do not, tell user and abort overlay run.
5. Put overlay command string into reverse polish notation. Check boolean logic for errors at same time.
6. Get the attribute merge "rules" from the user. These rules tell the attribute processor which attributes from which maps are to be passed along with the overlay process.
7. Validate the attribute rules by checking the field names against the attribute file. If there is an error, tell the user and let them change the field names.
8. Return to overlay processor.

3.4.1.4 User Interface

1. Please enter new map name
2. Please enter overlay command string or file of command input
3. Do you wish to retain attributes for map n?[Y]
4. If yes, enter field names of attributes to pass (CR=all).

Repeat 3 and 4 for all active maps specified in 2.

There are a number of boolean operations that are required to support the overlay process. These are:

M1 AND M2	(Intersection)
M1 OR M2	(Union)
M1 NOT M2	(Relative Difference)
NOT M1	(Single map Not)
M1 M/D	(Merge/Dissolve for Single Map)
M1 PASTE M2	(M2 AND NOT M1 OR M2 This is an update function)

The user would specify these primitive rules to control the complete overlay process. Please note that these are either single map or pairwise map operations. Complex overlays would be

accomplished by specifying a sequence of these primitives as well as the attribute operations that are associated with them. However, the use can hold the attribute processing in abeyance to be performed as a post process. In this case, the software automatically creates a file of feature ID's related to the input features.

A sample input command for the overlay process might be

```
MAP3 AND MAP4 FOR RESULT
Attributes MAP3 = ALL
Attributes MAP4 = HI,LOW,DIFFUSE
RESULT OR MAPS FOR FINAL MAP
Attribute RESULT = ALL
Attribute MAPS   = ALL
DONE
```

In this case, active map 3 is intersected with active map 4. All attributes for 3 or keep. Only attributes HI, LOW, and DIFFUSE are kept for map 4. The intermediate result is stored in a file called RESULT. A union operation is then performed between RESULT and active map 5. All attributes are passed through. The final map is stored in a dataset called FINALMAP. Intermediate results may be deleted or saved.

3.4.2 Pre-process Edge Data

3.4.2.1 Description

In order to optimize the physical overlay process and to have it operate closer to N*log(n) time rather than of N^2 time, the edge data must be preprocessed to determine candidates. The first step is to determine which edges fall in the geographic area of interest. The second step is to determine candidates for a major edge. This list of candidates is then passed to the intersection processor. Figure 3.11 is the structure diagram for the pre-processor.

3.4.2.2 Interface Information

1. MBR of current working geographic window
2. Does edge fall in geographic window? Yes or no. If yes, return Edge ID
3. Error return
4. Edge MBR
5. List of minor candidates for "major" edge.

3.4.2.3 Software Flow

Consider that we are overlaying two maps, map A and map B. Edges in map A are called major edges. Edges in map B are called minor edges.

1. Get a major edge.
2. Check to see if it falls in the current working geographic window.
3. If major edge falls in geographic window, determine which minor edges could possibly intersect it.
4. Scan minor edges, using the fast index file and banding to determine possible candidates. The banding tells you immediately if there is correspondence in X and Y. If there is correspondence in Y, then check edge MBRs for actual correspondence in X.*
5. Write major edge ID and all minor edge ID's to a temporary file.
6. Repeat 1 through 5 for all major edges.
7. Return to overlay processor.

* Please note that this step is modified by the type of operation. If the operation is an OR, keep all edges but also must still build candidate set. If the operation is relative difference (A AND NOT B), then keep all of A edges but only B candidates.

3.4.3 Intersection Processor

3.4.3.1 Description

Once candidates edges have been determined, then an intersection procedure must be executed to determine which edges actually intersect which edges and the geographic location of these intersections. The node, coordinate, and edge files are required for this procedure. In addition, a temporary file is required to store the intersection list. This temporary file has the following format:

WORD	BYTE	# BYTES	DESCRIPTION
1	1-4	4	Edge ID
2	5-8	4	Segment ID
3	9-12	4	Node ID
4	13-16	4	Other Edge ID

This structure is suggested by Franklin, 1983.

Each intersection point causes two records with these components to be created:

a) The number of the first edge that this intersection is on,
b) the number of the segment of this edge where the intersection is,
c) the node number of the intersection point, and
d) the number of the other segment on the other edge.

The second record stores the intersection from the point of view of the other edge. For example, see figure 3 which shows map A (solid lines) overlaid on map B (dashed lines). Ni, Ci, and Pi mean node #i, edge #i, and polygon #i, respectively. (Ni) represents a new node that is formed where two segments intersect. The intersection data structure for this map is

listed in table 2. In this, each edge's segments are numbered from one on up, going from the lower numbered node. Please note that all node ID's and edge ID's are unique.

3.4.3.2 Interface Information

1. Logical unit numbers for node files
2. Logical unit numbers for node-edge file
3. Logical unit numbers for node coordinate file
4. Error return
5. Start node ID for major
6. End node ID for major
7. Node coordinates for major
8. Edge coordinates for major
9. Start node ID for minor
10. End node ID for minor
11. Node coordinates for minor
12. Edge coordinates for minor
13. Intersection information

3.4.3.3 Software Flow

1. Initialize the intersection data structure described above.
2. Grab the major edge ID and the list of candidate minor edge ID's from the temporary file created by the preprocessor.
3. Grab the major edge.
4. Enter intersection loop, processing each minor edge against the major edge.
5. Find all the intersections of segments of edge A with segments of edge B. For some fast algorithms, see footnote. We suggest a routine called ARCINT, which exists in MOSS now. As each intersection point is found, enter it into the node table returning the new node number, and put two new records into the list of intersections. In the example of figure 3.13, there are four such intersections, nodes 4 to 8. It may happen that an edge intersects a node. In this case, an existing node number is returned from the node table. As before,if there is insufficient main memory to store all the nodes, the new nodes can be uniquely identified in a two step

---

Footnotes

Jon L. Bentley and Thomas A. Ottmann. "Algorithms for Reporting and Counting Geometric Intersections", IEEE Trans. Comput. C-28 (9), 643-647, (September 1979).

Jon L. Bentley and Derick Wood. "An Optimal Worst Case Algorithm for Reporting Intersections of Rectangles", IEEE Trans. Comput. C-29 (7), 571-576, (July 1980).

Kevin Q. Brown. "Comments on 'Algorithms for Reporting and Counting Geometric Intersections'", IEEE Trans. Comput. C-30 (2), 147-148, (February 1981).

process using sequential files. Also the list of intersection can actually be a sequential file.

6. Sort this list of intersections by the first two components, i.e. edge number and segment number. This will give us all the intersections of each edge in order along it. For example, edge 3 has intersections at nodes 5 and 7. As An external sort requiring little memory can be used.

3.4.4 Create new Arc/Node Map

3.4.4.1 Description

Once the intersection processor has completed the edge intersection process, the new map entry to the database must be built in preparation for verification, polygon formation, and attribute assignment. Figure 3.14 depicts this process.

3.4.4.2 Interface Information

1. Logical unit number for edge file
2. Logical unit number for coordinate file
3. Edge ID
4. Returned edge coordinates
5. Returned node ID's
6. Error return
7. New start/stop node for new edge
8. New edge coordinates
9. Number of coordinates

3.4.4.3 Software Flow

(Cut the old edges to make the new edges.) Make a single pass simultaneously through the list of edges and the sorted list of intersections. Cut each edge at each intersection of that edge with edges of the other map. Thus if an edge has N intersections, there will be N+1 new edges. Delete the old edge and insert the

new edges. Insert the numbers of the two new edges into the lists of adjacent edges of the relevant nodes. After this step, we have all the nodes and edges of the new overlaid map. This requires that only one edge be in memory at a time. In the example, edge 3 is split into three new edges: from node 1 to 7, from 7 to 5, and from 5 to 2. The first new edge has one segment, and the latter two, two segments each. Attribute keys for edges are written to an ancillary file for further processing. Attribute information comes from the feature file.

3.4.5 Polygon Formation

3.4.5.1 Description

This process, described earlier in 3.3, takes the edge, node, and coordinate data and builds the feature and chain files so that point, line, and area features exist in the database. This is the polygon formation processor.

3.4.6 Attribute Integration

3.4.6.1 Description

This processor determines what attribute information is associated with each of the new features built in the polygon formation process. It then updates the feature attribute keys so that the new feature either logically points to the appropriate set of existing attribute rows in the RDBMS or can build a new table of attributes based on the attribute rules specified by the user in the user input processor for overlay. Figure 3.15 depicts this procedure. The temporary file consists of one record each for each new edge. Each record contains the edge ID, Left, Center, Right attribute ID's for the major edge and Left, Center, and Right attribute ID's for the minor edge.

3.4.6.2 Interface Information

1 = Logical unit number for TEMP file of Edge ID's
2 = Error return
3 = Feature ID
4 = Logical unit for feature file
5 = Logical unit for chain file
6 = Logical unit for edge file
7 = List of Edges associated with feature
8 = List of attribute keys for a feature
9 = Attribute "passage" rules

3.4.6.3 Software Flow

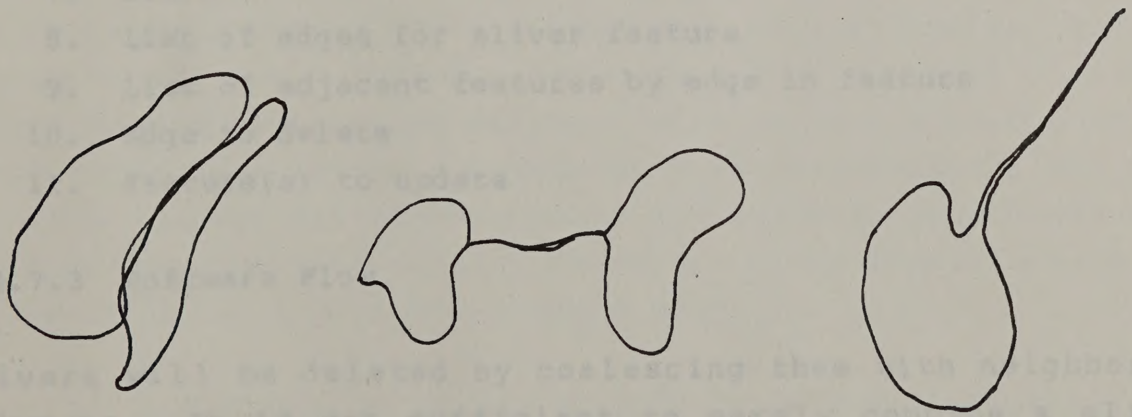
1. Open temp file. This file contains new edge ID's, and left, center, and right attribute keys for major and minor edges.
2. Perform an ascending sort using the edge ID as the sort key.
3. Grab a feature record from the new map.
4. Use keys in feature record to access the feature chain file.
5. Get a list of edges used to form the feature from the chain file.
6. Loop on the edge ID's, doing a binary search into the temp file for each edge in the new feature. Extract the correct attribute keys from the temp file.
7. Delete duplicate keys.
8. Pass the list of keys and the attribute rule set to the attribute merge processor to actually create a new row (or set of rows) for a feature.
9. Perform 3 through 8 until all features processed.
10. Return to overlay processor.

3.4.7 Sliver Removal

3.4.7.1 Description

Edges from input map A almost coinciding with parts of edges of map B may occur. For example, if map A is the state boundaries of America, and map B is the county boundaries, then edges defining part of the New York - Pennslyvania border will also occur on the county map as part of Erie county. The two edges will not have the same endpoints, but will approximately overlay each other for part of their length. Many small "sliver" polygons will result from overlaying these two maps. The more accurately the maps are digitized, the more slivers will result. Before giving a method for removing slivers, we must first characterize a sliver. One possibility would be any polygon feature whose area is smaller than a certain value. Another way is to determine the polygon's smallest "diameter" or the closest pair of parallel lines that enclose it. A third way is to use an area versus perimeter ratio. Figure 3.16 graphically portrays the sliver processor.

The figure below show various cases iin in which sliver removal may be desired.



However, sliver removal is an option that could be invoked at any point in an analysis stream, not just as part of the overlay

process. As a matter of fact, in certain cases, slivers may be significant and should not be removed. The 3 d$\overset{e}{i}$finitions given above for determining whether a feature is a sliver do not consider the importance of attributes or the accuracy of the input data. Therefore, the solution proposed in this documment is not a generic feature generalization algorithm but rather addresses only the very simple case of spatial slivers. Attribute precedence and accuracy are not considered. For example, coordinate data collected by first order survey techniques would never be moved or coalesced in neighboring features. Soils, timber, or wetlands would be moved to it and visa-versa. However, at some point in the feature, an expert system based on coordinate accuracy and attribute considerations could easily be multiplexed into the algorithm given here.

3.4.7.2 Interface Information

1. Feature key
2. Logical unit for feature file
3. Logical unit for edge file
4. Logical unit for coordinate file
5. Logical unit for chain file
6. Feature that is a sliver based on user definition
7. Error
8. List of edges for sliver feature
9. List of adjacent features by edge in feature
10. Edge to delete
11. Feature(s) to update

3.4.7.3 Software Flow

Slivers will be deleted by coalescing them with neighboring polygons. It is not sufficient to merely combine a sliver polygon with one of its neighbors since then it is possible for several sliver polygons to coalesce to form a non-sliver polygon. Thus, the algorithm for removing sliver polygons by

coalescing them with their neighbors is:

1. For each polygon, calculate its measure of "sliverness", be it area, smallest diameter, or whatever.

2. For each sliver polygon, find the longest edge bordering it that has a non-sliver polygon on the other side. Coalesce the two polygons by deleting this edge and renumbering the adjacent polygon fields of the other chains adjacent to the former sliver polygon.

3. Repeat step 2 for as long as there is a sliver polygon adjacent to a non-sliver polygon. Due to the coalescing procedure, a sliver polygon that is originally adjacent only to other slivers will eventually be made adjacent to a non-sliver feature.

4. If this whole component is nothing but slivers tell the user and ask if it should be deleted entirely.

3.4.8 Merge/Dissolve

3.4.8.1 Description

Closely related to sliver removal is a procedure called merge/dissolve. Sliver removal is a special case of merge/dissolve in which features with certain characteristics, such as minimum area, are removed from the dataset. The more classic type of dissolve is based on attribute considerations. Within the attribute domain, there are three possible cases to invoke a dissolve procedure. These are:

1. Attributes on either side of an edge are identical due to data entry error, certain cases arising from zone generation or overlay procedures, or attribute reclassification.

2. Attributes that are based on a hierarchial classification scheme, such as the Anderson Land Use Classification or the Cowardin Wetlands Classification. In this case, users may wish to move from level 2 to level 1 in the hiearchy and dissolve (or drop out) all edges that belong to level 2 in the hierarchy and work on with level 1 features.

3. By specifying a series of decision rules based on the attributes. For example, dissolve all edges (or features) that have features with attributes [sagebrush and deer habitat] on one side and [pinyon juniper and deer habitat] on the other side. This is not reclassification because we are not creating attribute sets. We are only dissolving edges based on attribute decision rules.

In all cases, dissolve requires some form of input of decision rules, even if it is as simple as if A=B, then dissolve. Figure 3.17 is the structure diagram for merge dissolve.

3.4.8.2 Interface Information

1. Work space.
2. # attribute rules.
3. Matrix of attribute rules.
4. Error from user input.
5. Logical unit number for attributes file.
6. List of edge ID's to retain after attribute filter.
7. Error from filter process.
8. New map name.
9. Error returns from map creation process.
10. List of errors from feature formation process.
11. Logical unit number for new attributes file.
12. Logical unit numbers for new maps.

3.4.8.3 Software Flow

1. Get attribute rules for dissolve process.
2. Get new map name.
3. Validate new map name.
4. Validate attribute rules by checking field names and field values against the source maps attribute file.
5. Run a filter process against all edges in map file and check against attribute rules. If the rules are satisfied, do not carry edge to new map. This sequence is:
   a. Get an edge record (this is a sequential pass through the edge file).
   b. Extract the left and right feature ID's.
   c. Grab the relevent feature records.
   d. Extract attribute ID's from feature record.
   e. Grab attribute data.
   f. Check attribute data against attribute rules.
   g. If rules not met, save edge ID.
   h. Go to next edge.
6. Use saved edge ID's to build a new map entry in the database. Use old edge, node, header, and coordinate records for this process.
7. Run verification/polygon formation process to build spatial structure.
8. Invoke attribute processor (if desired) to add attributes to new map.
9. Done.

3.4.8.4 User Interface

1. Enter new map name.
2. Enter attribute rules in form:

   Side 1 =
   Side 2 =

   If only wish to drop equalities, enter return. To

terminate input sequence, enter return.

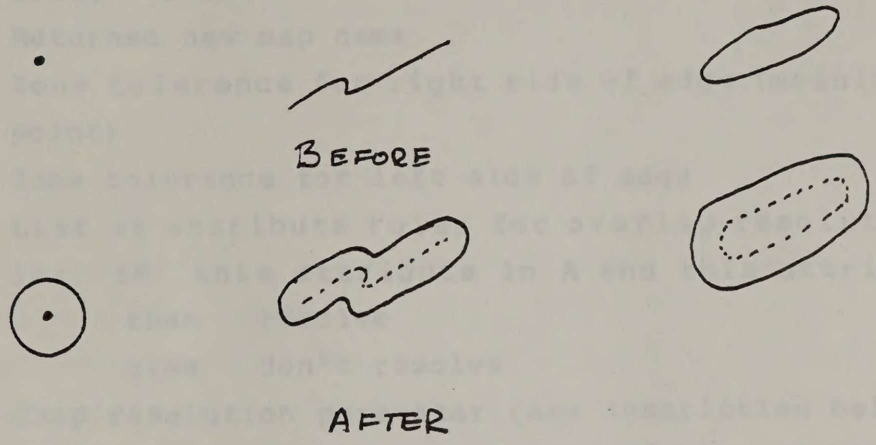
Sample

```
SIDE 1 = TYPE=SAGE, HABITAT=MULE
SIDE 2 = TYPE=PINYON, HABITAT=MULE
<RETURN>
```

3.5 Zone Generation

3.5.1 Description

Zone generation is the process of taking a point, line, or area, feature and creating a new set of polygon features based on generating profile offsets. For example:



Once the zones are created, then the overlay processor is invoked to resolve overlaps between features generated through the zone generation process. The actual overlap resolution is in part controlled by attribute rules. Once the overlay process is invoked, then the algorithmic procedure becomes the same as in the overlay processor (please see section 3.4).

The zone generation process contains three main modules: zone user interface, zone generation, and preparation for input to the overlay processor.

3.5.2 Zone User Interface

3.5.2.1 Description

The zone user interface module collect user input required to drive the zone generation processor. This module would also

validate user input. Figure 3.18 depicts the structure of the zone user interface.

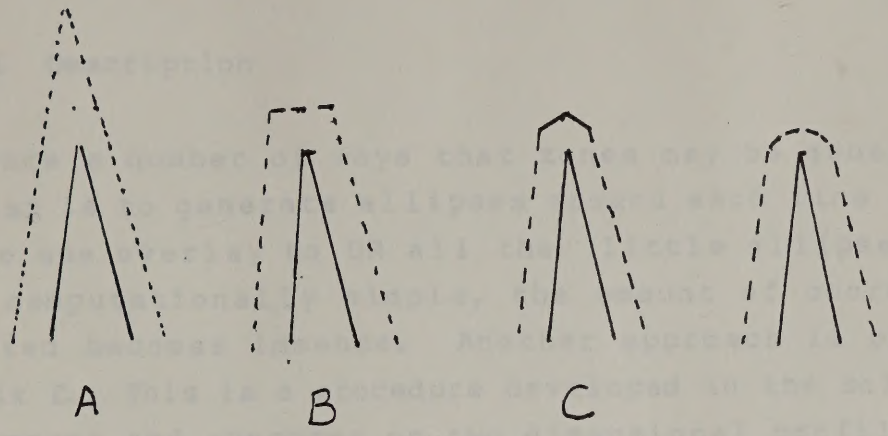
3.5.2.2 Interface Information

1. Logical unit for map names file
2. Returned characteristics of input map
3. Error return
4. Returned new map name
5. Zone tolerance for right side of edge (meaningless for point)
6. Zone tolerance for left side of edge
7. List of attribute rules for overlap resolution. Form is: IF this attribute in A and this attribute in B,
   then resolve
   else don't resolve
8. Cusp resolution parameter (see description below)
9. Number of coordinates to insert for cusp option D.

3.5.2.3 Software Flow

1. Get new map name from user
2. Validate new map name
3. Get old map name from user
4. Validate old map name
5. Get zone tolerance. User specifies units using current MOSS convention. Note that a lineal may have differential zone widths on either side.
6. Get the attribute rules for overlap resolution. This is a set of presedence rules (which don't necessarily have to be supplied) to let the overlay processor know which feature overlays must be resolved. For example, if feature A is type code 2 and feature B is type code 2, then perform a union operation between the two features. If A is 2 and B is 3, then A takes precedence, and so on.

7. Get cusp resolution specification. This tells the software how to resolve sharp corner situations during the zone generation process.



In A, distance T is greater than the zone tolerance. This will happen in all cases where the angle Q is greater than 45 degrees. Therefore, the software must do something to resolve this corner rounding situation. B is a straight cut across. C is the filleted V approach, and D is smooth corner rounding. D generates additional coordinate data.

3.5.2.4 User Interface

1. Please enter old map name
2. Please enter new map name
3. Please enter zone distance(s) [M for miles, F for feet, I for inches, K for kilometers, N for Nautical miles. Default is meters]
4. Please enter attribute precedence rules in Form

   IF A = STRING AND B = STRING, THEN A>B
   IF A = STRING AND B = STRING, THEN A=B
   IF A = STRING AND B = STRING, THEN A<B
   IF A = STRING AND B = STRING, THEN A=B
5. Please enter code for properly handling cusps (default is type C)

6. (If cusp type D) Enter number of coordinates to define cusp.

3.5.3 Zone Generation Process

3.5.3.1 Description

There are a number of ways that zones may be generated. The simplist is to generate ellipses around each line segment and then to use overlay to OR all the little ellipses together. While computationally simple, the amount of coordinate data generated becomes immense. Another approach is presented in Appendix E. This is a procedure developed in the solid modeling environment and operates on two dimensional profiles using B-splines and polynomials. Our original investigation suggested that it solves the general case. However, this solution is complex and is overkill for the simpler two dimensional case in which an overlay processor is also available. Therefore, we recommend that an intermediate approach be used for zone generation. The more complex algorithmic approaches should be addressed at a future date.

The algorithmic approach we suggest is:

1. Take a line segment from an edge.
2. Compute lines parallel but offset from the line segment (user input controls whether side 1, side 2, or both sides).
3. Clip parallel lines at endpoints of line segment.
4. Compute angle between current line segment and next line segment.
5. Compute bisector line that passes through the coordinate between two segments and through middle of interior and exterior angles.
6. If interior, clip parallel line.
7. If exterior, use cusp logic to insert coordinates from end of parallel line to bisector line.
8. Go to next segment.

Figure 3.19 depicts the zone generation process.

3.5.3.2 Interface Information

1. Logical unit for fast file for input map
2. Returned Feature ID if falls in window
3. Error Return
4. Number of edges for feature
5. Edge ID's for feature
6. Logical unit number of Feature file for input map
7. Logical unit number of Chain file for input map
8. Single edge ID
9. Logical unit number for Edge file for input map
10. Logical unit number for coordinate file for input map
11. Coordinates for input edge
12. Zone tolerance on left
13. Zone tolerance on right
14. Cusp specification
15. Returned coordinates for zone about edge
16. # coordinate pairs in zone
17. Logical unit for new edge file
18. Logical unit for new node file
19. Logical unit for new coordinate file

3.5.3.3 Software Flow

1. Grab Feature MBR from fast file
2. Check against Geographic Window. If in, continue. Otherwise, go back to 1
3. Get list of edge ID's for feature from chain file
4. Get coordinates for edge from coordinate file
5. Call zone generation module. The interface is the same regardless of the algorithm employed.
6. Add the new edge, node, and coordinate data into the new map database
7. Repeat 1 through 6 until all features are exhausted
8. Return to zone processor

3.5.4 Overlay Processor

When zone generation is complete, the necessary temp files are built and the overlay processor (less the user interface) is invoked. See section 3.4 for this discussion.

3.6 Polygon to Arc

3.6.1 Description

The Polygon to arc/node convertor takes a set of polygons in a single map and creates a set of nodes and edges that can be verified to create a true arc/node data structure environment. The Polygon to arc convertor has 4 logical processing modules: 1) user interface, 2) create a pre-processed file of major/minor correspondence (as in overlay), and 4) invoke the overlay processor.

3.6.2 User Interface

3.6.2.1 Description

The user interface for the polygon to arc process is shown in Figure 3.20.

3.6.2.2 Interface Information

1. Logical unit number to MOSS map names file
2. # Features in old MOSS polygon map
3. # Data type for old MOSS polygon map
4. Error Return
5. Fuzzy tolerance for melding almost coincident common lines

3.6.2.3 Software Flow

1. Get old MOSS map name for polygon formatted file
2. Validate old name against map names file
3. If match, get features and data type
4. Get new MOSS name from user
5. Check against map names file
6. Get fuzzy tolerance for handling near coincidence cases. In full polygon files, lines that are supposed to be coincident may not be exactly coincident due to digitizing error or coordinate storage precision. If lines that are coincident in reality are not coincident in the data, then the edges must be coalesced. This is the sliver removal procedure defined in section 3.4

3.6.2.4 User Interface

1. Please enter old MOSS map name
2. Please enter new MOSS map name
3. Enter "fuzzy" tolerance for sliver removal

3.6.3 Create Candidate File

3.6.3.1 Description

As with the overlay procedure, the next step is to determine "candidates" for the arc extraction procedure. This process involves chosing a major polygon from the old MOSS map and comparing it to all the other polygons in the same map. If MBR's overlap, then we have a candidate. This process is repeated for all polygons in the map, or $N^2 - N$ comparisons. For point or line data, this step is not required. Point and line maps can be turned directly into arc/node maps without going through the overlay processor. However, the verification process would have to be run on point/line maps to properly build the chain and feature files. Figure 3.21 depicts the pre-precessor function.

3.6.3.2 Interface Information

1. Logical unit number for DESCRIBE.FA
2. Major feature MBR
3. Major feature ID
4. Error return
5. Minor feature MBR
6. Minor feature ID
7. Overlap flag - true if features may be coincident, false otherwise
8. Logical unit number for pre-processor file

3.6.3.3 Software Flow

1. Open old MOSS fast file. Map must be active
2. Grab major MBR and feature ID
3. Grab minor feature MBR and feature ID
4. Insert major feature ID into preprocessor file. This preprocessor file has the same format as in the overlay processor
5. Compare major and minor MBRs. If there is overlap, then the minor is a candidate. If not, go back to step 3
6. If candidate, add minor feature ID to pre-processor file
7. Repeat steps 2 through 6 until all polygon features are processed

3.6.4 Invoke Overlay Processor

The final step is to enter the overlay processor at the point where two edges are passed to the intersection processor. From then on, the flow and processing is exactly the same as for the overlay processor.

3.7 Vector to Raster

3.7.1 Description

Vector to raster will require no new software. The only changes will be how the process will interface with new data structure. This is described in Section 4.

3.8 Edge Intersection

3.8.1 Description

Edge intersection is the process of comparing two edges for intersection points. The normal procedure is to compare all line segments in edge A against all line segments in edge B. This simplistic approach is slow. Additionally, all the special cases make what should be a simple process a lengthy, time consuming procedure. The special cases are:

1. Line segments parallel but not coincident
2. Line segments almost parallel that met at infinity
3. Line segments that should be coincident but are not
4. Line segments that are either vertical or horizontal
5. The endpoint if A fall on B or visa-versa

The code required to handle these cases is lengthy.

The problem of efficiency is handled in the proposed database structure by breaking edges every 64 coordinate pairs. Each edge is stored as a separate edge entity in the database. The worst case comparison is for an edge of 64 coordinate pairs versus an edge of 64 coordinate pairs, or 4096 comparisons.

The main problem then becomes one of bookkeeping: how do you keep track off all intersections for 2 edges and build the new arc/node map? The procedure discussed in the overlay section handles this problem.

Figure 3.22 has the structure of this process.

3.8.2 Interface Information

1 = X,Y coordinates for major edge
2 = Returned x,y coordinate of line segment end for major
3 = Returned x,y coordinate of line segment end for major
4 = X,Y coordinates for minor edge
5 = Returned x,y coordinate of line segment end for minor
6 = Returned x,y coordinate of line segment end for minor
7 = Flag - time = continue intersection process - false - get next minor segment
8 = Parameters for equation of line, major segment
9 = Parameters for equation of line, minor segment
10 = Intersection code
11 = Returned x coordinate of intersection (if exists)
12 = Returned y coordinate of intersection (if exists)

3.8.3 Software Flow

1. Get major segment from major edge
2. Get minor segment from minor edge
3. Do MBR's of major and minor segment overlap? If no, back to step 2
4. Check for vertical major segment. If vertical, jump to step (sub) 11
5. Check for vertical minor segment. If vertical, jump to step (sub) 15
6. Compute slopes of major and minor
7. Check slopes for parallel segments. If parallel, jump to step (sub) 17
8. Compute intersection based on geometry
9. Adjust computed intersection if fuzzy tolerance being used
10. Jump to step 22
11. If major vertical, check for parallel. If parallel, jump

to step (sub) 17
12. Compute intersection(s)
13. Adjust computed intersection if fuzzy overlay used
14. Jump to step 22
15. If minor is vertical, compute intersection
16. Adjust intersection if fuzzy overlay being used
17. If parallel, check following
18. Is end of major a point on minor?
19. Is end of minor a point on major?
20. Are the colinear?
21. Compute intersection for points above
22. Is geometrically computed intersection within both segment? If no, jump back to 2
23. Is intersection at beginning or end of segment? set codes
24. Process next minor segment against major until all minor segments processed
25. Process next major segment
26. Exit edge intersection process

3.9 DBMS Shell

3.9.1 Description

The DBMS shell is a set of subroutines that interfaces (and protect) MOSS with the attributes data manager. The shell is a set of processing modules, each performing a specific function, that would have to be recorded for each new RDBMS used. The important point in using a shell is that the applications software (MOSS) does not have to be modified. The interface modules are:

1. Open the attributes database
2. Validate a field (key)
3. Get a row of values
4. Put a row of values

5. Get a value from a row
6. Put a value into a row
7. Delete a row
8. Add a row
9. Change a value in a row
10. Create a new dataset
11. Delete a dataset
12. Boolean search

These routines already exist in MOSS. The question becomes how these should be modified to accomodate the new shell structure. This is discussed in implementation. This section will simply describe the information flow.

3.9.2 Open Attribute Database

3.9.2.1 Description

This module allows the application software to open the attributes database. This module will also load the attributes database schema information into memory. Basic error checking will also occur here. Figure 3.23 portrays the structure.

3.9.2.2 Interface Information

1. Returned database name. This name is created through software based on knowledge stored in the database
2. Logical unit number for attribute database
3. Number of attributes in database
4. Number of rows in database
5. Error - could not open database
6. Schema - includes key, type, length, and location

3.9.2.3 Software Flow

1. Create database name based on sheet name or theme name

2. Open database and read # rows and # attributes
3. Read logical schema into incore array (for virtual)
4. Pass information back to applications program

3.9.3 Validate and Find a Key

3.9.3.1 Description

This modules takes a field key supplied by the user or the application module and checks its validity (does it exist or not). If it does exist, return the type, length, and location. Figure 3.24 shows the structure.

3.9.3.2 Interface Information

1 = Key to check on
2 = Flag   True = key exists, False = key does not exist
3 = Error code
4 = Field type
5 = Field type
6 = Field location

3.9.3.3 Software Flow

1. Get key
2. Scan incore schema using sequential pass and string compare to find match
3. If no match, pass error back up to application program
4. If match, extract type, length, and location from schema
5. Return to application program

3.9.4 Get a Row of Values

3.9.4.1 Description

This routine returns a set of values (row) associated with a feature. The unique feature ID is used as the key into the DBMS. Figure 3.25 depicts this structure.

3.9.4.2 Interface Information

1. Attribute key
2. Error return
3. Make appropriate call to database to return a row (do not parse the row). This is a row of data.

3.9.4.3 Software Flow

1. Validate attribute key and reformat if necesssary for DBMS call
2. Make get request to database
3. Validate get
4. Return

3.9.5 Put Row of Values into Attributes Database

3.9.5.1 Description

This module acts exactly the same as the GET module except that it puts, or stores, a row of values into the attributes DBMS. Since it is the same, no structure chart is given. Please see subroutine description for DBPROW for more information.

3.9.6 Get a Value From a Row

3.9.6.1 Description

This module retrieves a single value, or field, from a row of attribute information for a given feature. Figure 3.26 has the structure.

3.9.6.2 Interface Information

0 = Row of attribute data
1 = Field key
2 = Field type
3 = Field lenght
4 = Field location
5 = Error - field not found
6 = Parsed value in character format
7 = Parsed value in correct format (Integer, Real, Character, Date)

3.9.6.3 Software Flow

1. Call CHKKEY to validate key and return type, length, and location
2. Parse field out of row of data
3. Convert to proper format

3.9.7 Put a Value into a Row of Attribute Data

3.9.7.1 Description

This module allows the applications software to place a value into a particular field in a row of attribute data. This routine could be used for initial population of the database or for update (change) of existing data in a row. See Figure 3.27.

3.9.7.2 Interface Information

1. Field key to replace
2. Row of attribute data
3. Field type
4. Field length

5. Field location
6. Error return - invalid field
7. Field value to insert (put)

3.9.7.3 Software Flow

1. Validate field key (CHKKEY)
2. Return length, type, and location
3. Convert from internal format to character string
4. Parse string back into row of attribute data
5. Return to application program

3.9.8 Delete a Row of Attribute Data

3.9.8.1 Description

This module deletes a row of attribute data from a relation.

3.9.8.2 Interface Information

1 = Attribute key (to row of data). This comes from the feature file
2 = Relation number
3 = Error return
4 = Relation name

3.9.8.3 Software Flow

1. Make sure relation pointer is pointing to proper relation
2. Set row pointer to correct row in table. Use the attribute key to do this
3. Delete the row using a RDBMS utility (in MOSS, under the current environment this is done by deleting the

feature. The attributes file is not touched)

4. Return to application module

3.9.9 Add a Row

3.9.9.1 Description

This module adds a row to a table. Please see section 3.9.5.

3.9.10 Change a Value in a Row

3.9.10.1 Description

This module allows for updating/changing a field value in a row of attributes. Please see section 3.7.

3.9.11 Create a New Dataset

3.9.11.1 Description

This module will allow the database administrator to create a new dataset. The existing MOSS Attribute database program or the RIM CREATE A NEW DATABASE module may be used to perform this function.

3.9.12 Delete an Attribute Database

3.9.12.1 Description

This module is used to delete an attributes database. This function is currently supported by MOSS. Only the calls must be changed. This is discussed in the implementation section.

3.9.13 Boolean Searches

3.9.13.1 Description

This module accepts a user attribute query as input, passes it into the required format for the DBMS, has the DBMS do the search, and returns a list of feature keys for those features who's attributes satisfied the search criteria. See Figure 3.29.

3.9.13.2 Interface Information

1. Query in current MOSS format
2. Number of keys
3. Number of relationals
4. Number of Boolean operators
5. Key stack
6. Relational stack
7. Relational Values
8. Boolean stack
9. Query in DBMS format
10. Results of query
11. Sorted on feature ID (list of feature ID's selected)
12. Relation number and name
13. Error

3.9.13.3 Software Flow

1. Make sure relation pointer is up to date.
2. Convert the MOSS attribute query into reverse polish. MOSS subroutine RPN does this
3. Convert from RPN into DBMS query format. Use DBMS and schema rules to do this. The schema is used to validate field names and type
4. Call the DBMS search routine
5. Sort the result on the feature ID
6. Return list of feature ID's
7. Return to application program

4.0 Implementation

4.1 Introduction

Previous sections described data structures, information flow, structure diagrams, and algorithmic approaches for arc/node processing and attribute manipulation. This section describes specific implementation strategies. Before the specific areas of enhancement described in Section 3 are addressed, we will begin with a discussion of how an arc/node structure can be 'slipped in' as another data type in MOSS.

4.2 Incorporating a new data type in MOSS. (Phase I)

Currently, MOSS has a data type code 4 that has been reserved for an arc/node implementation. Every vector processing routine in MOSS checks data type. Without enhancing any functional capability, how can an arc/node data type be added?

The first task is to implement the arc/node database primitives. A list of these primitives is found in Appendix F. A test harness would also have to be written to allow for thorough testing of these primitives before incorporation into MOSS actually begins.

The second task is to upgrade the existing spatial DBMS commands to use the new arc/node primitives. A list of routines and required changes follows:

BLDHED -	This routine builds a MOSS map header. It needs to be enhanced to handle the new fields described in section 2.3.10.
CORDXY -	Add check for arc/node data type. Call feature build routine (FEABLD) if type arc/node. The feature build routine returns a set of coordinates and islands in the traditional MOSS format.

CORDPQ - Add a check for arc/node data and call feature build routine (FEABLD).

GETSUB - This routine would have to be modified to check for data type. If the data type is arc/node, then call the DBMS routine that returns a value for a field for a particular feature (this is the current MOSS GETATT routine).

IGETIS - This routine would disappear. CORDXY and CORDPQ would be modified to return the island pointers.

GETWND - This routine would have to be modified to check for arc/node. If arc/node, extract the feature MBR from the map fast file (subroutine GETMBR).

GETNDX - This routine would be replaced by calls to GETFAST. It would then disappear as a MOSS routine.

PRNHED - This routine prints out a MOSS map header. It will have to be modified to accommodate the new header structure described in section 2.3.10.

These are the only MOSS primitives that would have to be changed to handle the new spatial data structure (attributes are a different matter and are discussed separately).

The third task is to implement the new DBMS shell. This shell contains a number of primitives that are described in Appendix F. As with the spatial primitives, these shell routines should be implemented in a stand-alone basis with a test harness to ensure that they function properly. The common blocks required for the DBMS shell are described in Appendix F.

At the end of these three tasks, MOSS will be fully operational in regard to: 1) using the old polygon files, and 2) reading data in arc/node format from the database. However, the SELECT command using the new attribute structure would not be operant. In addition, the PLOT command must be modified; it reads coordinate data directly from disk and does not use CORDXY or CORDPQ. Also at this stage, any analytical functions that

operate from the arc/node structure will create full polygon maps in the old MOSS format as output. Thus, even after the three above tasks have been completed, MOSS as an arc/node processing system is not to the point for release in a production environment even though all the building blocks would be in place. We are now at the point where we can follow the outline of functional capabilities given in Section 3 as the basis for an implementation strategy.

4.3 Attribute Manipulation (Phase I)

This section describes how the attribute DBMS shell is incorporated into MOSS. We assume that a shell has been written for the target DBMS. We suggest for an initial implementation that the existing multiple attribute structure be used. In this way, when Phase I enhancements become operational, users will have a complete attributes capability without having to get a proprietary RDBMS.

The 4th task is to change or replace the existing multiple attributes subroutine interfaces. This is a relatively straightforward process. The routines (and the routines that call them) along with the required changes are as follows:

1. Database open: Update subroutine INTIAL to call the DPOPN subroutine as well as the routine RELCOM,which loads the logical schema into memory. Common block /SCHEMA/ must be added. INTIAL is called by:

    PR ATTRIBUTE.
    PR COMPUTE.FR
    PR LEGEND.FR
    PR QUERY.FR
    PR REPORT.FR
    SU CMFILE
    SU DESC
    SU MASEAR

SU SDESCR
SU SEARCH
SU TABARA

Please note that PR means program and SU means subroutine.

2. Get a row of data for a feature: Currently, ATTMVG performs this function. The ATTMVG call should be replaced with the call to DBGROW. Currently ATTMVG is called by only POLYMVG. Also QUERY could use this routine. QUERY currently does an inline "get row" equivalent.

3. Put a row of data into database for a feature. No current MOSS code performs this function. However, the new processors (Overlay, Attribute edit, and so on) will require this capability. The DBMS shell routine is DBPROW.

4. Get a field value from a row. This function is currently serviced by the GETAT and GETMAF. These routines should be integrated into one routine that performs their current function by organizing the call to the DBMS shell routine DBGVAR. GETMAF is called by QUERY and SAVWRK. GETAT is called by:

   PR COMPUTE
   PR LEGEND.FR
   PR SAVWRK
   SU GRIDPT
   SU MAEVAL
   SU PLOTA
   SU SDESCR
   SU TABARA

5. Put a field value into a row: This function is currently served by the MOSS routines PUTAT and PUTMAF. These routines should be consolidated into one routine that re-arranges MOSS parameters into the correct call to the shell routine DBPVAR. PUTAT is used by the ATTRIBUTE program. PUTMAF is used by

QUERY and by SAVWRK.

6. Delete a row of attribute data: This function is not currently used in MOSS. The DBMS shell routine is DBRDEL.

7. List schema definition: This function is currently done by subroutine HEADER. This should be changed to a call to the shell routine DBSLST. HEADER is currently used by ATTRIBUTE and LISTAT.

8. List values for field: This function is currently serviced by subroutine LISTAT. LISTAT must be upgraded to use the appropriate DBMS shell routines. LISTAT is called by ATTRIBUTE and DESC.

9. Boolean search of attribute data: This function is performed by the MOSS routine MASEAR. This routine would have to be modified slightly for the multiple attributes file structure proposed in this document. This is because the schema is in memory and not on disk. To interface to another DBMS, this routine would have to be completely re-written and becomes the DBMS shell routine DBSEAR.

10. Merge two rows of attribute information. This function is currently performed by MERGAT. MERGAT will have to be upgraded to use the shell routines for getting and putting rows and field values from within a row. MERGAT is currently called by the ATTRIBUTE program.

11. Check a field name: This routine is already serviced by subroutine RETRVL. This routine should be replaced with CHKKEY.

This completes the current attributes primitives that will have to be changed. We can now turn our attention to the MOSS commands, or processors, that use the primitives and discuss the

changes that must be made to them. This is Task 5.

1. LIST: The list command allows the user to either list the schema (header) or the values for a particular field for all features in a map sheet. Since these functions are serviced by the HEADER and LISTAT routines, these changes are already accomodated. The LIST SUBJECTS option will have to be changed. A subroutine called SUBDMP will have to be modified to list just the major attribute key. The attribute key will become the equivalent of a subject in the arc/node environment.

2. SELECT: Select allows users to search over one field in the multiple attributes file. This is not a boolean search but users like it because it is easy to use. BSEARCH allows the user to input complex boolean search strings. The SELECT attribute option should invoke the current BSEARCH and the BSEARCH command should be deleted. This way, only one attribute query format is required.

3. ATTDESCRIBE: The function allows the user to build a new schema for a multiple attributes file. This program must be upgraded to :
   a) allow users to edit a schema,
   b) list a schema,
   c) add information on field relationships,
   d) store valid ranges of values for numbers.

   In a RDBMS environment, this program would not exist because the DBMS schema editor would be used.

4. ATTRIBUTE: This is the main database administrator program for the multiple attributes file. It allows the user to create, edit, list, and in general maintain an attributes database. This program would not be used in an RDBMS environment. This program would have to be enhanced to:
   a) use attribute keys more easily for sort and merge

   operations (required for map update, attribute file creation),
   - b) use the DBMS shell routines,
   - c) attribute validation on input,
   - d) allow row deletion/addition.

5. QUERY: This command dumps out all the attribute information associated with a feature identified with a crosshairs. This routine must check for type arc/node and use the new DBMS shell to perform the attributes interfacing.

6. EDITATT: This command allows the user to identify a feature with the crosshairs and change any field value. This is the same code as QUERY. By changing QUERY, EDITATT will also be changed.

7. CALCOMP: This command uses two routines for plotting attributes for hard copy output. CMFILE calls the database open routine. It also gets a list of keys from the user for the fields to plot. PLOTA gets the field value and plots it. These routines would have to be enhanced to properly use the DBMS shell.

8. LEGEND: This command has the option to plot attributes for features displayed on the CRT. This routine will have to be modified to use the correct DBMS shell routines (DBGVAL).

9. STATISTICS: This command has a descriptive statistics option that works with attributes. Subroutine SDESCR will have to be modified to use the correct DBMS shell routines (DBGVAL).

10. AREA/LENGTH/PERIMETER: These commands print summary statistics of area, length, or perimeter based on attribute classes. Subroutine TABARA will have to be modified to use the correct DBMS shell routines (DPOPN, DBGVAL).

11. REPORT: The report command allows the user to create simple attribute reports. This routine would not be used if an RDBMS approach was chosen. Otherwise, the INTIAL and GETAT calls will have to be changed to use the DBMS shell. Also, there is considerable bookkeeping in REPORT that keeps track of where attributes are in the database. These will have to be changed to use the stored logical schema.

12. COMPUTE: This command allows the user to perform spread sheet analysis on the attributes file. This routine will have to be upgraded to allow the creation of a new field using the DBMS shell. Also, the field GET and PUT routines will have to be upgraded to use the DBMS shell.

13. SAVWRK: SAVWRK creates a new MOSS map. It will also create a new attributes file based on a decision file passed in by the calling program. Subroutine SAVMAT does the work of merging attributes from one or two input maps into a new attributes file. This routine will have to be modified to make correct usage of the DBMS shell.

14. GRID: This command does point to grid interpolation. It can work off the attributes file. Thus, the CALL to INTIAL and GETAT are going to have to be changed to use the new DBMS shell routines.

15. POLYCELL and RASTERIZE: These commands do vector to raster conversion. All values can be extracted from the attributes file. The calls to INTIAL and GETAT must be changed to use the DBMS shell. Also, some inline code is going to have to be changed to use the DBMS shell. This inline code is prefaced with comments.

This completes all the upgrades/enhancements required to upgrade the existing attributes capabilities in MOSS to make effective use of the DBMS shell. All that remains is to implement

functional capabilities for attribute manipulation that do not exist.

We now must consider implementing several functional subroutines/processors that do not currently exist. The becomes the sixth task. These are:

1. Attribute Reclassification

If an RDBMS is being used, then this function should be performed by the user through the RDBMS. If the current MOSS multiple attributes approach is desired, then a new module to perform the attributes reclassification is required. The characteristics of this routine are described in section 3.1.1. To further clarify that discussion, there are two modes of attribute reclassification. One is to use existing attribute data as input to an algebraic expression to compute a new field value. In this environment, the user enters a new field name and description. The software creates a new field entry in the attributes table and then computes new values and adds them to the new field. The MOSS function COMPUTE already performs this process. Changes to COMPUTE have already been discussed.

The second reclassification is to enter a set of rules to completely redo all or part of the attributes database. This capability allows the user to change and reconfigure the attributes without changing the topological or spatial structures. This particular capability would have to be coded from scratch. The first step should be to completely specify the user interface. A strawman interface was given in 3.1.1.4. This should be reviewed by users to make sure that it is correct. Once the user interface is agreed on, then the driver and user interface software can be coded. The DBMS shell routines are used to get and put rows and field values.

2. Attribute Dump to Interchange File

At times, users wish to dump MOSS attributes information to an ASCII file that can be used as input to some other program or DBMS. This capability does not currently exist in MOSS. Logically, this capability should be integrated into the current MOSS EXPORT command. EXPORT should ask the user if they wish an attributes file to be dumped. If the answer is yes, then the user should be queried for the schema name. The schema is used to convert the internal attributes storage structure back out into the original ASCII input format. The DBMS shell routine called DUMPAT is called to perform this function.

4.4 Data Entry (Task seven)

The first subtask in Task seven is to upgrade the GENERATE command to accommodate the functional requirement for data entry specified in 3.2 as well as to integrate in the arc/node structure. The second subtask is to code and test an edge match procedure.

4.4.1 GENERATE Enhancement

The GENERATE command already provides for key and cursor entry. Cursor entry supports rectangles, circles, points, lines, and polygon features. The arc and ellipse feature entry capabilities must be added. We also strongly suggest that the user interface be re-worked to make user entry/control easier. The arc and ellipse generation routines should be integrated as subroutines. The interface to these routines should be generic so that they could be used in later MOSS enhancements or in COS3.

The next step for GENERATE is to add the arc/node interface. This step will cause some major changes to the software. However, this upgrade is not required until phase II of the evolution into a complete arc/node implementation for MOSS. Hence, it is not discussed here.

4.4.2 Edge Match

This is major new software module. The approach given in section 3.2.3 is suggested for the algorithmic sequence. The Edge Match module should be implemented as a stand alone program executable through the MOSS command structure and subroutine CTERP. The edit station should be a Tektronix compatible graphics CRT. A cursor is required. Color is preferable.

4.5 Verification/Polygon Formation

Task 8 is to upgrade the verification/polygon formation software used by AMS to use the new arc/node data structure. We also suggest that the HP-9000 version be used because 1) its already in FORTRAN 77 and 2) island assignment is as much as 10 times faster than in the Data General version.

The biggest obstacle to pursuing this implementation is that currently all record access to the database is embedded in the code. This includes calculation of record indexes. There are about 300 of these calls. These will have to be changed to the appropriate database primitive subroutine calls. The channel numbers used in the current read/write operations can be used to determine which files are actually being accessed. The routines that are currently used in verification are:

BMAP*
BTEST*
CFILL*
CMOVE*
DFLCT
EGSRT
ENPR
ESEGS
FRMID
INDX
ISRCH*
IZERO*
KROSS
MESS
NNPR
PARGS*
PAUSE*
PUTIX*
PUTIY*
IX*
IY*
RANGE
READF
WRITF
V1
V2
V3
V4

V5 V6 V7 V8
V10 V10AT V10ED V11
VCLR

The * routines do not need to be converted. In addition, routines V2, V4, and EGSRT are not required for the MOSS implementation.

The formation process builds the feature chain file and the feature file. The attribute processor would be executed to perform the attribute tagging for the newly formed features.

4.6 Arc/Node Overlay

Task 9 is the implementation of the arc/node overlay processor. This also includes the edge intersection algorithm. This is the largest task in the Phase I implementation. Whenever possible, legacy code should be used. An example of this is the edge intersection routine given in Appendix G. The algorithm approach given in section 3.4 should be followed. By the time the arc/node overlay processor Task is initiated, the DBMS shell and the database primitives will already have been implemented and tested. The overlay processor implementation breaks into the following sub-tasks:

4.6.1 User Interface

The user interface is described in section 3.4.1.

4.6.2 Edge Intersection

Edge intersection is described in section 3.8. A strawman routine is given in Appendix G. This routine should be interfaced to the run-time structure described in section 3.4.3. Then, very rigorious testing of this module is required to check it against all pathological cases. A standard data set should be used for this testing. The Bureau of Reclamation, Sacramento has a very good benchmark dataset that could be used for this function.

4.6.3 Preprocessor

Implement as described in section 3.4.2. Use Spatial DBMS primitives for database interfacing.

4.6.4 Arc/Node to Polygon Formation

Use module implemented and tested as described in section 4.4.

4.6.5 Attribute Information

Use the map attribute merge software described in sections 3.1.5 and 4.1.

4.6.6 Sliver Removal

This is new code. Use the approach described in section 3.4.7. We suggest using minimum distance between parallel lines as the approach to determine what is a sliver.

4.7 Zone Generation (Task 10)

The implementation of the zone generator is comprised of several sub-tasks.

4.7.1 User Interface

The first sub-task is to complete the user interface specification with input from the user community. Once this is completed, then the existing MOSS BUFCOM subroutine should be enhanced to allow for new or modified user inputs.

4.7.2 Zone Generation Algorithm

The algorithm suggested in section 3.5.3.1 should be implemented. An individual who is extremely strong in mathemetics and computer science should be used to implement this algorithm. This is the major work component in the zone generation sub-task. This software module should be thoroughly tested for point, line, and polygon information. Some pathological cases should be designed and used during the test procedures.

4.7.3 Invoke overlay processor

This sub-task builds the necessary input files so that the overlay processor can be invoked to resolve overlaps in the buffered features. This is discussed in section 3.5.4.

4.8 Polygon to Arc

Polygon to arc is Task 11. This is the procedure to break polygons apart into edges and nodes. A procedure was discussed in section 3.6. This is new code. The primary effort is to build a candidate file for the overlay processor. The two main sub-tasks are the user interface and the pre-processor to build the input file to the overlay processor. The polygon formation is performed by the formation processor. The attribute tagging is performed by the same processor as used by overlay.

4.9 Vector to Raster

This is Task 12. It simply requires that the POLYCELL and RASTERIZE commands recognize the arc/node data type. Since the existing MOSS primitives have been upgraded in Task 2 to pass back full polygons from the arc/node database, the majority of the effort is completed. However, the arc/node file must be integrated in so that no logical unit conflicts occur.

4.10 Miscellaneous Modules Required for Phase I Implementation. (Task 13)

In addition to the enhancements and new modules already discussed, there are a number of existing MOSS modules that must be upgraded to allow for a completely operational MOSS at the end of Phase I. These are now discussed.

4.10.1 AUDIT

The AUDIT command dumps information on a feature by feature basis in a report format. The information displayed in the table is stored in different locations in the arc/node database then in the current environment. Thus, the database extraction portions of AUDIT will have to be enhanced to work in the new arc/node environment. These fields are: primary attribute, perimeter, area/length, number of coordinate pairs, and number of features.

4.10.2 ADD

The ADD command currently works for full polygon data. A new module is required that takes arc/node data as input, adds the data into MOSS, runs the verification/polygon format processor, and then the attribute processor to build a new MOSS arc/node map. We suggest that the arc/node interchange format for MOSS be DLG 3 ASCII. There is already an AMS to DLG 3 ASCII module. ADD would be upgraded to create an arc/node map, allow the user to

specify arc/node input, and then invoke a routine that reads that format. A routine that reads DLG format already exists in the DLG2MOSS conversion package. The remainder of the software required for map add is available from previous tasks - polygon formation and attribute tagging.

4.10.3 DLG2MOSS

This routine will not be required for arc/node MOSS due to the enhancements to the ADD command.

4.10.4 DUMP

DUMP is a DBA/support routine that dumps in ASCII format, the exact contents of a MOSS map file. This module would be enhanced to dump chain, feature, node, edge, and coordinate files in arc/node MOSS. This routine should actually be upgraded as part of Task 2, since it is a very valuable debugging tool.

4.10.5 SIZE

SIZE is a retrieval function. It will have to be modified to retrieve the area/length field from the arc/node feature file as well as the old polygon feature header.

4.10.6 STATISTICS (DESCRIBE)

Statistics allows the user to generate descriptive statistics. The attributes interface has already been discussed. In addition, STATISTICS must be modified to extract area/length information from the correct location in an arc/node data structure.

4.10.7 STATUS

STATUS will have to be enhanced to print out system level summary

statistics for an arc/node map. This would include number of edges, number of nodes, number of features, number of coordinate pairs, and file lengths.

4.10.8 Merge/Dissolve

The final new module to incorporate into MOSS is the merge/dissolve processor. The attribute reclassify portion will have been completed as part of Task 6. The actual Merge/Dissolve module will be new software. The algorithm for this new software is described in section 3.4.8 as part of the overlay processor. Basically, a set of decision rules is used to determine which edges/nodes are based to the new map. Then the verification processor is invoked. In the current AMS environment, a program called MEDGE already performs this function.

4.11 Phase I Summary

This completes all the requirements for Phase I enhancement to MOSS. The system will have complete functional capability. However, the majority of the analytical commands will still create full polygon maps as results as opposed to arc/node maps. This brings us to Phase II.

4.12 Phase II

Phase II efforts will focus on the remainder of the analytical and data manipulation commands. The emphasis is for them to 1) work more directly from the arc/node data structure and 2) to create resulting maps in arc/node structure. These commands are:

CONTIGUITY: The CONTIGUITY function will realize substantial time savings in the arc/node environment. The current MOSS uses a full polygon compare. At times, this is very slow.

Using the edge file will speed this considerably.

CONTOUR: CONTOUR will create an arc/node file under this scenario. This will allow the verification processor to form polygons from the edges. This capability has been desired for some time.

DIVIDE: DIVIDE will now become a split/arc, re-verify procedure, much cleaner than the current polygon divide.

EDGE: EDGE will work directly from the attribute and edge files. This will cause the software to execute much more rapidly than in the current polygon environment in which edges are re-computed from contiguous polygon boundaries.

GCONTOUR: Same as CONTOUR.

LATTICE: No significant difference.

LPOVER: The new overlay processor will replace the functional capabilities of LPOVER. LPOVER will disappear.

MERGE: MERGE would work from arc/node files. The verification processor would have to be invoked to verify the new features, especially those that cross map sheet boundaries.

MOVELABEL: MOVELABEL will have to be changed to read from the arc/node structure as well as to read and write the centroid from the feature file.

POINTS: Write to arc/node data structure. No real improvement.

POLYMVG: Same changes as for RASTERIZE and POLYCELL. Also, will have to change interface to the attributes file.

PROJECTION: Change interface only to coordinate file, area,

length, and MBR fields. Should run at least twice as fast since coordinate data only being scanned once.

PROXIMITY: Work with arc/node structure. Should run at least twice as fast due to way in which coordinate data are stored.

SAVE: See MERGE

SMOOTH: Work from arc/node file. Also, since edges will be processed only once, no slivers will be created as occurs in the polygon environment. Create a new map and invoke verification processor.

TRANSLATE: Work with arc/node structure. Should run at least once as fast since coordinate data stored only once.

WEED: Work with arc/node data. Will operate correctly since coordinate data traversed only once. In MOSS, it is now traversed twice - once in each direction in terms of coordinate data. Will also run much more quickly.

EXPORT: Read from arc/node structure and output in DLG3 modified format.

It should be noted that Phase II could actually begin after Task 6 has been completed in Phase I.

4.13 Phase III

Phase III would focus on adopting arc/node structure and DBMS interface for the remaining functions in MOSS. This would also include removing redundant polygon processing software. The conclusion is that users must convert all their old MOSS polygon maps into the arc/node structure.

Phase 3 will also provide an opportunity for redefining the user

interface or perhaps placing interface shells on top of MOSS so that the user's view of the software and the data can change but the actual software does not change.

5.0 Level of Effort

5.1 Introduction

This section describes level of effort for tasks identified in section 4. Estimates do not include computer time. Estimates include final design, user and systems documentation, and testing. Assumption is that scope of work is as specified in Sections 3 and 4. Finalize design is primarily the user interface and subroutine specification.

TASK 1 Arc/Node Database primitives

• Finalize arc/node primitive subroutine specifications	1 man month
• Code primitives	2 man months
• Test primitives	.5 man month

TASK 2 Upgrade existing MOSS spatial DBMS primitives

• Finalize design considerations	.5 man month
• Perform upgrade	2 man months
• Perform test	.5 man month

TASK 3 DBMS Shell

• Choose RDBMS (or MOSS) and finalize DBMS shell design	1 man month
• Code	2 man months
• Perform test	.5 man month

TASK 4 Change MOSS attribute interface at the primitive level

• Finalize design	.5 man month
• Code changes	1.5 man months
• Test changes	.5 man month

TASK 5 Upgrade MOSS drivers to use new DBMS Shell routines

. Finalize design	1 man month
. Code upgrades	1 man month
. Test upgrades	1 man month

TASK 6 Add new attributes capabilities

. Finalize design	.5 man month
. Code Enhancements	1.5 man months
. Test Enhancements	.5 man month

TASK 7 Data Entry

. Finalize design	.5 man month
. Code upgrade	1.5 man months
. Test upgrade	.5 man month

TASK 8 Verification/Polygon Formation

. Code enhanced version	2 man months
. Test enhanced version	1 man month

TASK 9 Arc/node overlay, including edge intersection and merge/dissolve

. Finalize design	.5 man month
. Code overlay processors	3.5 man months
. Test overlay	1 man month

TASK 10 Zone Generation

. Finalize design	.5 man month
. Code module	2 man months
. Test	1 man month

TASK 11 Polygon to arc

Code module	2 man months
Test module	1 man month

TASK 12 Vector to raster

Code changes	.5 man month
Test	1 man month

TASK 13 Complete Phase I

ADD	1 man month
DUMP	.5 man month
DLG2MOSS	1 man month
SIZE	1 man week
AUDIT	.5 man month
STATUS	1 man week
STATISTICS	1 man month

5.2 Phase II

CONTIGUITY	1 man month
CONTOUR	1 man month
GCONTOUR	.5 man month
DIVIDE	.5 man month
EDGE	1 man month
LATTICE	.5 man month
MERGE	1 man month
MOVELABEL	.5 man month
POINTS	.5 man month
POLYMVG	1 man month
PROJECTION	1 man month
PROXIMITY	1 man month
SAVE	.5 man month

SMOOTH	.5 man month
TRANSLATE	.5 man month
WEED	.5 man month
EXPORT	1 man month

5.3 Phase III

Estimated level of effort 6 man months

APPENDIX A

1.0 INTRODUCTION

This document is a description of the database management capabilities currently in MOSS. The database management capabilities in MOSS may be thought of as focusing in three main areas: map library functions, spatial data storage and manipulations, and attribute storage and manipulations. Each of these major categories is discussed in this document.

2.0 MAP LIBRARY FUNCTIONS

The highest level in the MOSS database belongs to the map library functions. These functions are designed to allow the MOSS user to determine which maps are in the database and the characteristics of the maps. Some of these functions are part of the main MOSS program; others are utility functions available through the MOSS.UTIL menu. Functionally, they are:

- List all the maps in a MOSS database in alphabetical order along with their data type.
- List only derived maps.
- List only original data.
- List only those map names that match a search template; i.e., all maps that begin with the letters AB.
- Create a map index file, sorted by map name, that shows map name, scale, map description, and number of features for all maps in a MOSS database. This file may be saved or printed on a line printer.
- Create a graphic map index showing quad (or map) boundaries at any scale. This index map may be displayed on the CRT or on a hardcopy plotter.
- List those maps that have been archived.
- List the header for an individual map sheet.
- List the cartographic symbolizations for a map sheet.
- List the projection/coordinate storage transformations for a map sheet.
- List the color assignments for features on a map sheet.
- Print out the number of features, coordinate pairs and islands in a map sheet.
- Delete a map sheet from the database.
- Change the name of a map sheet in the database.
- Change the status of a map sheet in the database.
- Edit a map sheet header.

The available functions for performing operations at the map

library level are constantly being expanded. It should be noted that the majority of the map library functions work exactly the same for vector or raster data.

3.0 SPATIAL STORAGE AND MANIPULATIONS

This section describes those capabilities in MOSS related to the storage, retrieval, and manipulation of the spatial portions of a MOSS database.

- Delete a feature from a map sheet.
- Dump a MOSS map sheet entry in ASCII format to the terminal or to a line printer. This is not the MOSS import/export format but an actual field by field dump of the database.
- Select a set of features from a map based on size or length criteria.
- Select a set of features from a map based on their proximity to a point (give me all the lakes within 5 miles of Sioux Falls, S.D.).
- Select a set of features from a map based on their proximity to another set of features from another map (give me all the lakes that are within .5 miles of a paved road).
- Select a set of features based on juxtaposition (give me all the lakes that are next to a paved road).
- Select a set of features based on elevation (if z values exist) (give me all the houses that lie between 1000 and 1200 feet.)
- Change the cartographic symbolizations for a map.
- Change the color assignments for a map.
- Point to a set of displayed features on a CRT and have their characteristics printed out at the terminal.
- Change the projection/coordinate system of the coordinate data in a map.
- Convert from vector to raster format.
- Rotate, translate, and scale map sheets already in the database.

4.0 ATTRIBUTE STORAGE AND MANIPULATION

This section discusses the capabilities related to attribute storage and manipulation. This particular set of capabilities has undergone significant changes over the last year.

- Create an attribute descriptor file that contains the format of an input attributes file (the ASCII file). This file is used for performing multi-variable batch adds of the attribute information. This file can be edited to reflect changes in the input format.
- Export attribute files for entry into other database management systems.
- Store integer (whole), real (floating point), and character attributes.
- Update one field (variable) in the attributes database.
- Update, sort, and merge the attribute data to reflect a change in the spatial/cartographic data (i.e., give me all features of attribute type pinyon juniper).
- Perform complex boolean attribute retrievals (i.e., give me all features in a map that have an assessed tax value of between 1000 and 2000 dollars and have been re-assessed since 1980).
- Edit an individual value for an individual field by pointing to the displayed feature on the CRT with the graphics cursor and making the necessary changes.
- Point to a feature with the graphics cursor and have all the attribute information for that feature dumped to the screen.
- Use the attributes information as the basis for coding cell values in the vector to raster conversion process.
- Use the attribute information as the basis for interpolating a continuous surface (digitial elevation model). For example, select a salinity field for bathymetry data and interpolate a salinity surface.
- Display attributes data on the CRT associated with the cartographic data.

- Set cartographic symbolizations based on attribute.
- Create area, length, perimeter and/or frequency reports for a map based on the attributes.
- Create an attribute report with up to 6 columns of attribute information. The output may be disposed to a line printer.
- Prepare a Statistical Summary of an attribute field to show frequency, min, max, range, mean, standard deviation, variance, Total.
- Use the attributes to create a new attribute which is stored in the data base through the use of algebraic expressions. For example, use feature area along with population to compute density values. This is akin to a spread-sheet utility.
- Create new attribute and feature sets based on the results of vector overlay of files containing multiple attributes.
- Create multiple attribute files associated with individual grid cells based on the rasterization of vector data with associated multiple attributes.

5.0 DATABASE COMMANDS

There are currently 39 commands which invoke the database management functions. These are:

1. ADD	Adds import/export format vector maps into the database
2. ARCHIVE	Prepares maps for removal from the database to tape
3. AREA	Areas aggregated by attribute descriptors
4. ASSIGN	Initial assignment and edit of cartographic symbols, scaling and colors
5. ATT ADD	For adding attributes to an existing map
6. ATT BROWSE	Browsing the header information for attribute files
7. ATT DESCRIBE	Create an attribute descriptor file
8. ATT EDIT	Disk file and key board entry for update of attribute fields
9. ATT RESEQ	Attribute resequencing for adding new information from external sources
10. AUDIT	Prepares summary report of all features in a map
11. BROWSE	Enables the user to list descriptors of each map in the database and its status i.e. active or archived
12. BSEARCH	BOOLEAN REETRIEVAL (LE, GE, GT, LT, EQ, NE, AND, OR, NOT) for multiple fields
13. CALCOMP	Generation of multicolor hard copy plots for CALCOMP, ZETA, VERSATEC (monochrome and color), H-P, HOUSTON INSTRUMENTS, GERBER
14. COMPUTE	"Spread Sheet" functions and add computed variables to the attribute file
15. DATABTEST	Adds imported map names to the index files and is used to change status of maps in the database
16. DEARCHIVE	Reloads archived maps into the database
17. DELETE	Deletes a map from the database
18. DESCRIBE	Enables the user to list any or all attributes

	(bounded or unbounded)
19. EDITATT	Interactive edit of attributes on feature by feature basis
20. EXPORT	Export function for Spatial and attribute information
21. GENERATE	Creates a new vector map interactively from a graphics terminal
22. GRID	Use point attribute data to interpolate a continuous surface, for contouring and profiling
23. LABEL	Allows for the entry or edit of header and projection information associated with every map
24. LEGEND	For interactive soft copy labelling
25. LENGTH	Length aggregated by attribute descriptors
26. LIST	Lists cartographic symbol and color assignments and projection information
27. MAPIDX	Construct an index map of maps in the database or project
28. MERGE	Combines two or more vector maps into a single map
29. OPEN	Allows access to other databases
30. OVERLAY	Polygon overlay (merge) of maps with multiple attributes
31. POLYMVG	Convert vector data with multiple attributes to multivariable raster files
32. QUERYATT	Interactive determination of attributes
33. REPORT	Prepares 6 column soft and hard copy reports
34. RENAME	Renames entries in the database
35. SAMPLE	Performs random sampling of features in a vector map
36. SAVE	Saves an active map as a new map in the database
37. SELECT	a) ALL ATTRIBUTES BASED ON ID OR KEY b) RANGE BASED ON ID OR KEY AND LOWER AND UPPER BOUNDS
38. STATISTIC's	DESCRIBE, min, max, range, mean, standard deviation, variance, total, frequency
39. TRANSLATE	Rotate, translate, scale map sheets

APPENDIX B

FILE NAME: TYPE 1,2,3,5,11,12,13 POINT LINE POLYGON VECTOR MAPS MULTIPLE ATTRIBUTE FILE. THIS FILES PRESENCE IS INDICATED BY WORD 76 OF RECORD ONE IN THE MAP HEADER. THE .AT HEADER RECORD IS FOLLOWED BY ATTRIBUTE RECORDS 2-202, FOLLOWED BY RECORD 203 WHICH IS RESERVED, WHICH IS FOLLOWED BY THE ATTRIBUTE DATA RECORDS.

RECORD TYPE: ATTRIBUTE HEADER RECORD RECORD ONE OF .AT

WORD #	BYTE #	# BYTES	TYPE	DESCRIPTION
1	1-2	2	INTE	# OF ATTRIBUTES (1-200)
2	3-4	2	INTE	# OF FEATURES IN MAP
3-128	5-256	252		RESERVED

FILE NAME: TYPE 1,2,3,5,11,12,13 POINT LINE POLYGON VECTOR MAPS MULTIPLE ATTRIBUTE FILE.

RECORD TYPE: ATTRIBUTE RECORD RECORD 2-202 OF .AT

WORD #	BYTE #	# BYTES	TYPE	DESCRIPTION
1	1-2	2	INTE	ATTRIBUTE ID...A SEQUENCE #
2-11	3-22	20	CHAR	ATTRIBUTE KEY...10 CHARACTERS RIGHT JUSTIFIED ONE CHARACTER PER WORD
12-83	23-166	144	CHAR	ATTRIBUTE DESCRIPTION...72 CHARACTERS RIGHT JUSTIFIED ONE CHARACTER PER WORD
84	167-168	2	INTE	ATTRIBUTE TYPE...1-INTEGER 2-REAL 3-CHARACTER
85	169-170	2	INTE	ATTRIBUTE LENGTH IN WORDS INTERGER-1 REAL-2 CHARACTER<73
86	171-172	2	INTE	FIRST RECORD IN FILE CONTAINING ATTRIBUTE DATA FOR THIS ATTRIBUTE
87	173-174	2	INTE	# OF RECORDS REQUIRED TO STORE THE ATTRIBUTE DATA FOR THIS ATTRIBUTE (128/WORD 85)/RECORD 1 WORD 2
88-128	175-256	82		RESERVED

FILE NAME: TYPE 1,2,3,5,11,12,13 POINT LINE POLYGON VECTOR MAPS MULTIPLE ATTRIBUTE FILE.

RECORD TYPE: ATTRIBUTE RECORD RECORD 203 OF .AT

WORD #	BYTE #	# BYTES	TYPE	DESCRIPTION
1-128	1-256	256		RESERVED

FILE NAME: TYPE 1,2,3,5,11,12,13 POINT LINE POLYGON VECTOR MAPS MULTIPLE ATTRIBUTE FILE. THE ATTRIBUTE DATA VALUES FOR EACH FEATURE ARE STORED AND ACCESSED IN THE SAME SEQUENCE AS THE FEATURES. I.E. THE ATTRIBUTE DATA FOR THE FIFTH FEATURE IS IN THE FIFTH ATTRIBUTE DATA SLOT. ALL ATTRIBUTE DATA FOR ATTRIBUTE ONE IS FOLLOWED BY ALL ATTRIBUTE DATA FOR ATTRIBUTE TWO ETC. ONCE THE ATTRIBUTE FILE IS BUILT ADDITIONAL FEATURES CAN NOT BE ADDED, HOWEVER MORE ATTRIBUTES CAN BE ADDED.

RECORD TYPE: ATTRIBUTE DATA RECORDS   RECORD 204-N OF .AT

WORD #	BYTE #	# BYTES	TYPE	DESCRIPTION

IF WORD 84 OF ATTRIBUTE RECORD IS 1-INTEGER
THE INTEGER ATTRIBUTE DATA IS STORED ONE VALUE PER WORD. THE NUMBER OF VALUES PER RECORD IS 128. THE NUMBER OF RECORDS TO STORE THE DATA IS THE NUMBER OF FEATURES/128. THE REMAINDER OF THE LAST RECORD IS UNDEFINED.

IF WORD 84 OF ATTRIBUTE RECORD IS 2-REAL
THE REAL ATTRIBUTE DATA IS STORED ONE VALUE EVERY TWO WORDS. THE NUMBER OF VALUES PER RECORD IS 64. THE NUMBER OF RECORDS TO STORE THE DATA IS THE NUMBER OF FEATURES/64. THE REMAINDER OF THE LAST RECORD IS UNDEFINED.

IF WORD 84 OF ATTRIBUTE RECORD IS 3-CHARACTER
THE CHARACTER ATTRIBUTE DATA IS STORED ONE CHARACTER RIGHT JUSTIFIED PER WORD. THE CHARACTER FIELD IS NULL FILLED. THE CHARACTER FIELD DOES NOT CROSS RECORD BOUNDS AND AS A RESULT THE ENDS OF THE RECORDS ARE UNDEFINED AND THE END OF THE LAST RECORD IS UNDEFINED. THE RECORD WOULD ACCOMADATE 12 TEN CHARACTER FIELDS, 6 TWENTY CHARACTER FIELDS, 1 EIGHTY CHARACTER FIELD, ETC.

APPENDIX C

POLYGON FORMATION/VERIFICATION

Up to this stage of processing, the map has been registered, segments digitized and the intersections of segments found. Other than the fact that segments may share common junction points (nodes), they are essentially unrelated at this point. Polygon Formation/Verification is the process of tying together the segments associated with mapped areas. As implied by the title "Verification", an important by-product of this process is an error analysis of digitized features. Errors detected at this stage include incorrectly labelled segments or polygons, crossing segments, edge nodes not tied to, missing segments and other less obvious problems.

Beginning at one segment in the polygon, bounding segments are followed, based on angle relationships, until the entire perimeter is closed. Record numbers of each segment forming the polygon are copied to a polygon data file. Islands must also be distinguished at this stage and assigned to their respective "parents". Additionally, the ADS program must assign attributes to polygons.

AMS

Polygon Verification can be run in AMS in either interactive or batch modes. The interactive mode is controlled to a large

extent by the operator who must correct errors as they are detected prior to resumption of the program. Batch mode frees the user from constant attention by automatically collecting a list of errors that can be repaired at a later time.

AMS breaks down formation/verification into 7 primary tasks, each addressed by a separate subroutine and controlled by the VERF program. Typically, it is necessary to run a map through verification at least twice to ensure that all errors have been corrected. Prior to the initial run through the program, no polygons exist and all segments must be processed. Beyond the first run, however, many polygons will have been successfully formed and verified and require no further processing. The first task, V2, identifies all segments that have been modified since the most recent run through verification. Subsequently, when the program is "resumed" only those polygons associated with modified segments are cleared (deleted) and reformed/verified, thereby saving the computer resources necessary for verifying the entire map. However, a final run through of the program is still necessary to assure that any modifications to repair one error have not created other errors.

The second task, V3, generates several parameters for use later in polygon formation. The primary purpose of this routine is to calculate the area contributions of segments in the map. Each segment contributes a positive or negative area to a closed figure based on its azimuthal direction from the map origin

(geounit center). Generally, a segment digitized from west to east and south to north contributes positively while the opposite directions contribute negatively. The area is calculated as a summation of trapezoidal areas formed by internal segments and the axes of the origin (Figure 8.1). These area contributions are used primarily for classifying a polygon's islands or parents and are stored in the Segment Verification file($$16).

The V4 routine next sorts edge nodes clockwise from the map's northwest corner and simultaneously searches for edge nodes not linked to a segment. Such an error, perhaps resulting from an edge node extraction/edge match problem, will immediately discontinue the verification process and must be repaired before any further tasks can be initiated. The sorted format allows the program to more efficiently direct polygon formation when the map edge borders a polygon. Unlike ADS, the map boundary is not stored as a segment and requires a few more calculations to determine its location. A maximum of 600 edge nodes is dictated by the sorting array used, however, a map must have a minimum of 1 edge node (and connected segment) to insure that at least one polygon encloses the map's features. The necessary segment (whether labelled "null" or otherwise) ties the map boundary to the map interior.

The V5 routine handles the actual formation of polygons. The method used forms polygons, and determines the presence of islands; however islands are not yet assigned to parents. As

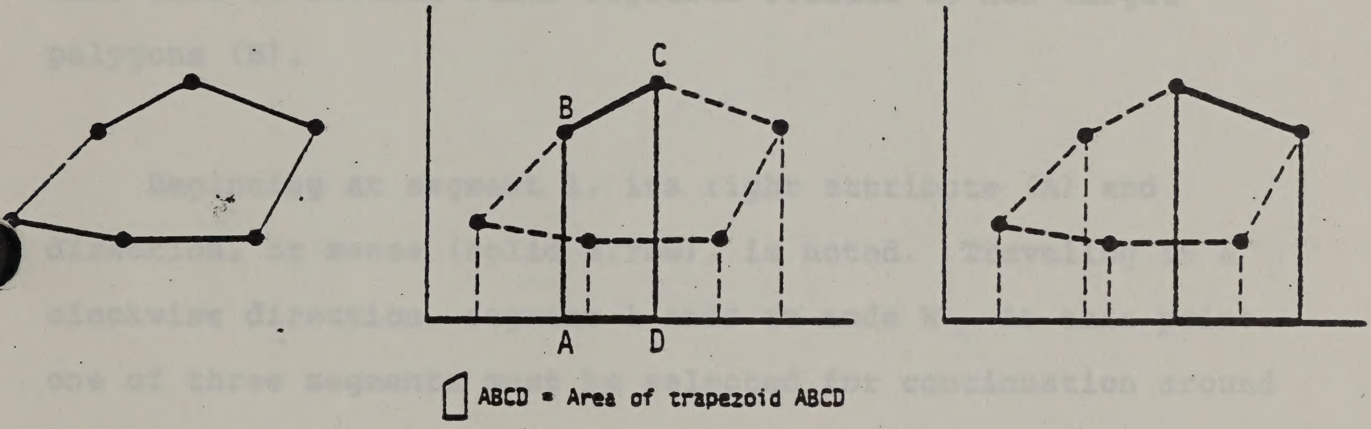

C
B
A
D
ABCD = Area of trapezoid ABCD

Figure 8.1: Calculation of segment/polygon area contributions; areas of all trapezoids are summed for a total

discussed earlier, AMS segments exhibit several characteristics. They begin at a node, travel in a direction (sense) with attributes to the left and right, and end at a node. By consistantly following "right" attributes, the entire map can be classified into a system of polygons and simultaneously verified by checking the consistancy of attributes within the polygon.

For this explanation, Figure 8.2 will be referred to for simplicity. This figure illustrates a target polygon (A) surrounded by 4 segments (1-4) tied to 4 nodes (W-Z) which are also tied to several other segments related to non-target polygons (B).

Beginning at segment 1, its right attribute (A) and direction, or sense (solid arrow), is noted. Traveling in a clockwise direction, segment 1 ends at node W. At this point, one of three segments must be selected for continuation around the polygon. The segment chosen is that which results in the minimum angle (taken counterclockwise) from the entering segment (1) to the exiting segment. In this case the minimum angle is formed by segment 1's intersection with 2. Taking into account the direction segment 2 was digitized, its right attribute (in a clockwise sense) is noted (A) and its ending node (X) examined for intersecting segments. The attributes are compared as the process continues, flagging errors if discrepancies exist. Segments are followed until the starting point (segment 1) is reached and the polygon is closed. Polygon B would next be

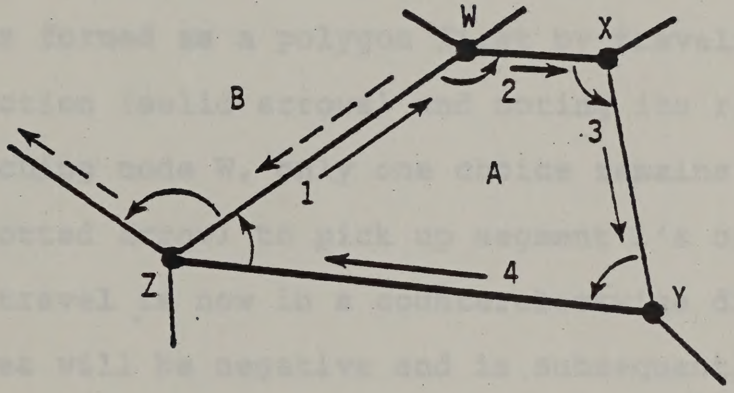

W
X
B
2
3
A
1
Z
4
Y

Figure 8.2: AMS polygon formation process (see text for explanation)

formed by reversing the direction of segment 1 (dotted arrow), noting its "right" attribute (B) and again travelling in a clockwise direction around the polygon (B).

As segments belonging to the polygon are found, the area contribution of each, calculated in V3, is summed, resulting in a total area encompassed by the polygon. This area value is stored with the polygon header record.

The special case of an island is illustrated in Figure 8.3 . Polygon (A) is bounded by one segment (1) closing on itself at node W. (A) is formed as a polygon first by traveling in a clockwise direction (solid arrows) and noting its right attribute (A). Upon reaching node W, only one choice remains and direction is reversed (dotted arrow) to pick up segment 1's other attribute (B). Because travel is now in a counterclockwise direction , its accumualted area will be negative and is subsequently classified as an island. Each segment is followed a third and final time to account for its center attribute which is labelled in the case of a linear feature.

The two most common types of errors detected at this stage are attribute discrepancies and cases where a segment has been used more than the alloted three times. Attribute errors can typically be traced to a mistake in entering the attribute prior to digitizing or to a problem of crossing segments near a node. The digitizing technique used in approaching a node will affect

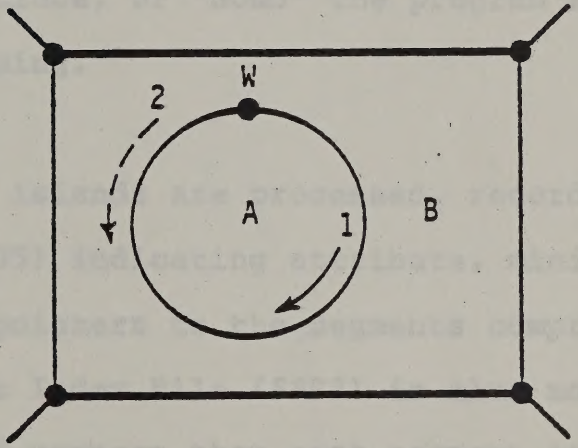


Figure 8.3: AMS island formation process (see text for explanation)

the frequency of crossing segments. Too many points sparked near the node may result in a problem illustrated below (Figure 8.4). In this case, the polygon being formed follows segment 1 to where it enters node Z. The next correct path should be segment 2, however because segment 3 exits at the least counter clockwise angle, it will be chosen, resulting in an attribute conflict. Such errors will be accumulated in a summary when using batch mode verification. However, as many of these errors are found in the same general region, polygon formation becomes confused and can cause a segment to be used more than three times. This type of error will discontinue, or "bomb" the program and requires repair prior to resuming.

As polygons and islands are processed, records are built in the polygon file ($$05) indicating attribute, minimum bounding rectangle, area and pointers to the segments comprising the polygon. The Segment Index File ($$03) is also modified to point to the polygon record numbers that each segment is related to. Two "holding" files, the Polygon Verification and Island Verification files ($$17 and $$18), respectively,are also created to store polygons and islands separately for island/parent assignment. This task, handled by V6, assigns the islands formed in V5 to their parent polygons. Any polygon or group of polygons totally enclosed by, and not connected to the parent is classified as an island and must be identified with its parent. This operation is necessary to retain the island/parent spatial

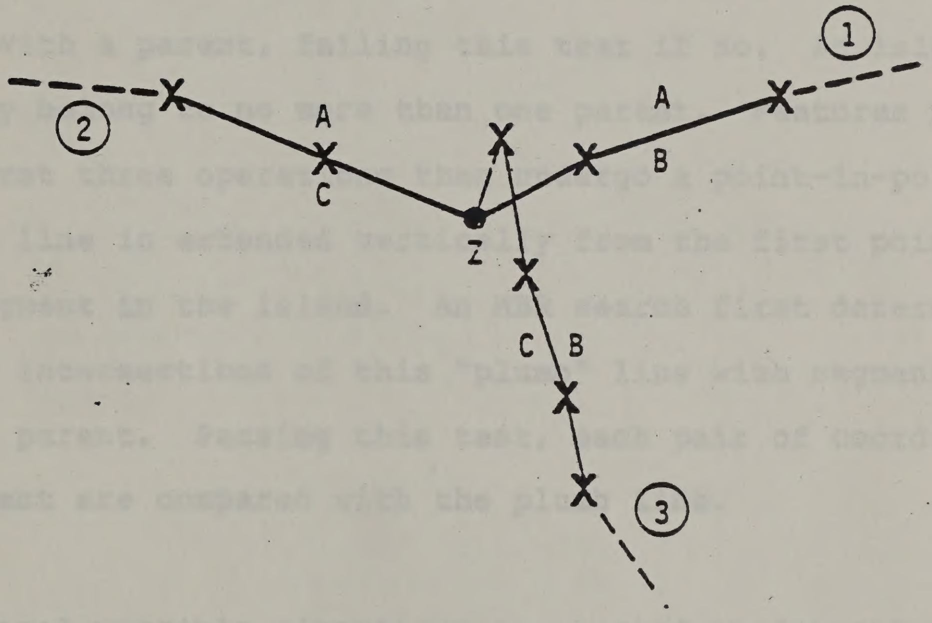

1
2
A
A
B
C
Z
C
B
3

Figure 8.4: Crossing segments causing polygon formation error in AMS (see text)

relationships primarily for the purpose of area computations and feature searching.

Three tests are initiated to determine the correct parent for an island. An MBR search, comparing each polygon in $$17 to each island in $$18, produces a list of possible parents. Next, the attributes of the segments comprising each possible parent and island are compared for continuity (the "outside" attribute of the island must match the "inside" attribute of the parent). The island is then checked to determine if it has already been matched with a parent, failing this test if so. An island can obviously belong to no more than one parent. Features passing these first three operations then undergo a point-in-polygon test. A line is extended vertically from the first point in the first segment in the island. An MBR search first determines possible intersections of this "plumb" line with segments of the possible parent. Passing this test, each pair of coordinates of the segment are compared with the plumb line.

Several possible situations are tested to determine an intersection with the plumb and for simplicity are illustrated in Figure 8.5. Each intersection of a segment with a polygon is tallied. An even summation indicates the island is outside the parent while an odd result indicates an island/parent relationship. An island and parent passing all these tests is then designated a "most likely" pairing. It is possible, of course, in the case of nested polygons, that an island may belong

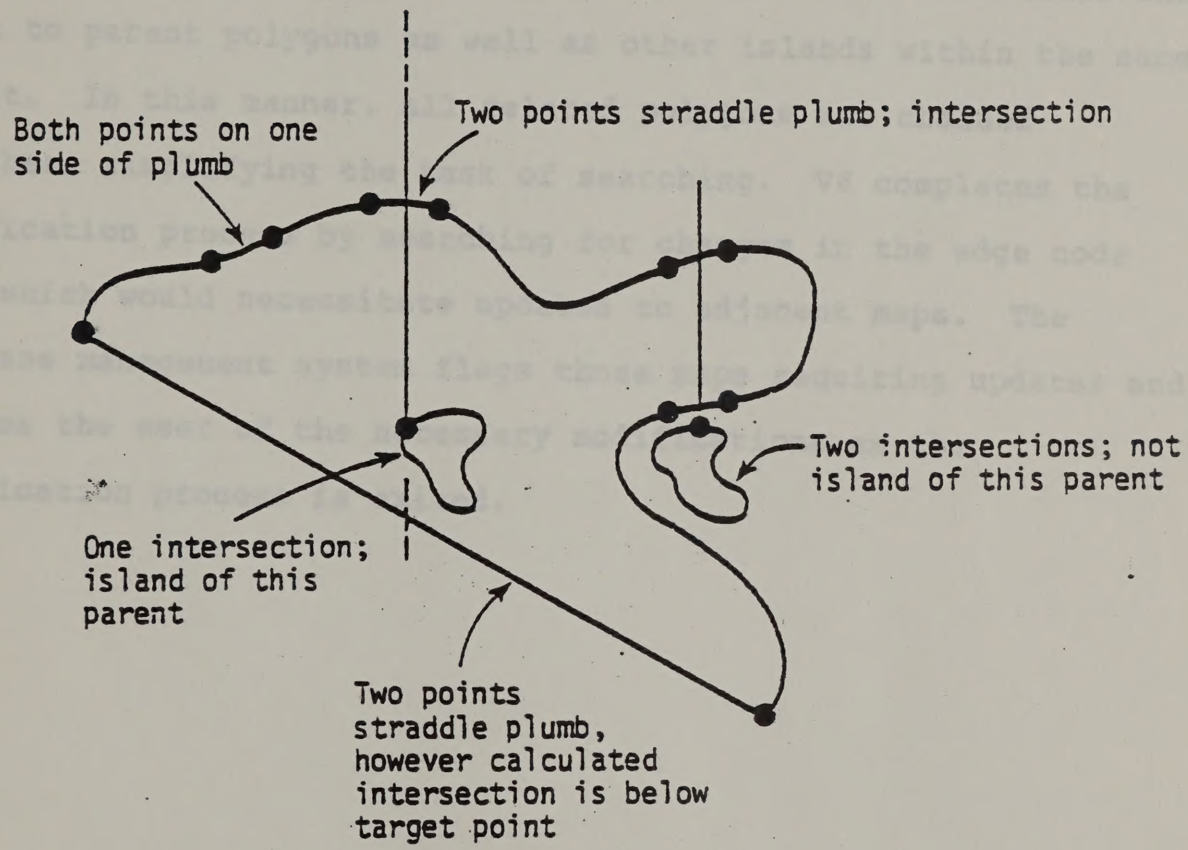

Two points straddle plumb; intersection
Both points on one side of plumb
Two intersections; not island of this parent
One intersection; island of this parent
Two points straddle plumb, however calculated intersection is below target point

Figure 8.5: Criteria used for assigning islands to parents

to two parents. Therefore, a final test identifies the parent having the most restrictive MBR so that only first generation islands/parents relationships are eventually recognized in the updated polygon file ($$05).

The sixth task, V7, follows-up island assignment by adjusting polygon records to indicate the presence of islands and point to parent polygons as well as other islands within the same parent. In this manner, all related polygons are chained together, simplifying the task of searching. V8 completes the verification process by searching for changes in the edge node file which would necessitate updates to adjacent maps. The database management system flags those maps requiring updates and informs the user of the necessary modifications as the verification process is exited.

ADS

APPENDIX E

An offset capability for planar curves and profiles in a solid modeler leads to the solution of some practical design problems.

Offsets of Two-Dimensional Profiles

Wayne Tiller and Eric G. Hanson
Structural Dynamics Research Corporation

There are a number of important uses of offset profiles in design and manufacturing. One example is mold design, where a draft angle is applied to a mold surface. In this procedure, a profile is offset slightly, the offset translated, and then a ruled surface formed from the profile to its offset. This allows the molded part to be easily extracted from the mold after the material has cooled.

At Structural Dynamics Research Corporation, we have implemented offset, two-dimensional profiles in the solid modeler Geomod.[1] We went through several phases in the development of this capability. First, we implemented a very primitive algorithm to allow us to create some offsets quickly and easily and thus get an idea of the problems we would encounter. We quickly learned that the general offset problem is very complex.

In view of that complexity, the second step was to decide how complete a solution we should attempt to implement; in other words, which of the more difficult offsets would be useful in a practical geometric modeling system? Our first thoughts were that most of the exotic cases are probably not really important. But, after working with our designers and studying a few practical design problems, we realized that many of the difficult cases arise very naturally in real design problems. (The offsets shown in Figure 14 are one example.) For this reason, we decided to implement a general and robust capability that can handle most of the offsets encountered in actual design situations.

The third step was the design and implementation of the necessary algorithms, a step heavily influenced by the geometry representation and system architecture of Geomod. Rather than a detailed treatment of specific algorithms, we give here an overview of our work.

Terminology

We use the following terminology:

- C denotes a curve in the xy-plane. C may be open or closed but may not intersect itself at any interior points. Curves are assumed to possess a well-defined tangent vector at each point. $C(t)$ denotes a parametrically defined curve, with parameter t, where $a \le t \le b$, $0 \le a < b$.
- Let T be the unit tangent vector to C at a point Q; i.e., $|T| = 1$. The unit normal to the curve C at Q is defined by $N = Z \times T$, where Z is the vector (0,0,1), and × denotes the vector cross product.
- Let $P = \{C^i\}$, $1 \le i \le n$, be a collection of curves such that the end point of C^i coincides with the start point of C^{i+1} for $1 \le i \le n-1$. Except for the end point linkages, we allow no pairwise intersections among the $\{C^i\}$. P is called a profile. Figure 1 shows examples of open and closed profiles.
- Let $P = \{C^i\}$ be a profile. Q_s^i denotes the start point of the ith curve of P, and Q_e^i the end point of the ith curve. See Figure 1. Q^i denotes an arbitrary point on the ith curve; T^i and N^i, arbitrary unit tangents and normals.
- Let d be a fixed, nonzero real number, and let C be a curve. The offset to C at the (negative or positive) distance d is the set of points in the xy-plane given by $O = C + d \cdot N$. Because O may have cusps and/or loops (Figure 2), it does not generally satisfy our above definition of a curve. For this reason, we call O simply an offset, not an offset curve. (A Geomod

0272-1716/84/0900-0036$01.00 © 1984 IEEE

user will usually want to use curves and offsets to construct surfaces, so he will want to remove all loops and perhaps to fillet out cusps. In a later section on loop removal, we will show how this is done when the underlying geometry representation is nonuniform rational B-splines. See also Tiller.[2])

[illegible] precisely define an offset of a profile is not possible [illegible]e unique normals do not necessarily exist at the junction points of the curves making up the profile. While this might discourage a mathematician, a designer usually has a specific idea of what the offset of his profile should look like. His idea, however, is sometimes rather subjective, and another designer might expect a different offset of the same profile. Figure 3, for example, shows a profile with three possible offsets, any one of which might be meaningful. Therefore, we do not attempt to define an offset, but rather we give a designer a number of capabilities and options to create offset profiles that are useful in his work.

We assume a working knowledge of both polynomial and rational B-spline curves, but for completeness we give the following basic definitions and terminology. (For more detail see Tiller,[2] Riesenfeld,[3] DeBoor,[4] and Versprille.[5])

A bounded, polynomial B-spline curve, $C(t)$, is defined by the following mapping from a bounded segment of one-dimensional parameter space into two-dimensional Euclidean space:

$$C(t) = \sum_{i=1}^{n} B_{i,k}(t)\, Q_i, \qquad (1)$$

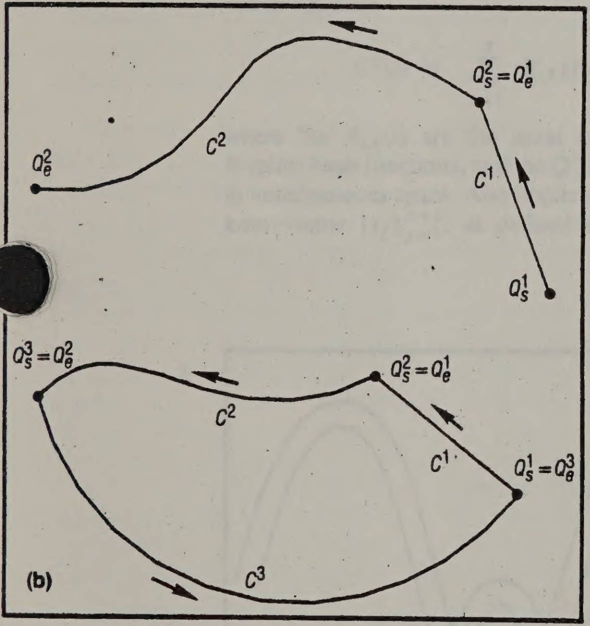


Figure 1. Open profile (a); closed profile (b).

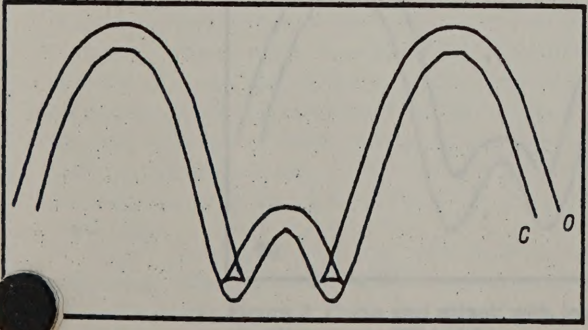


Figure 2. Offset containing four cusps and two loops.

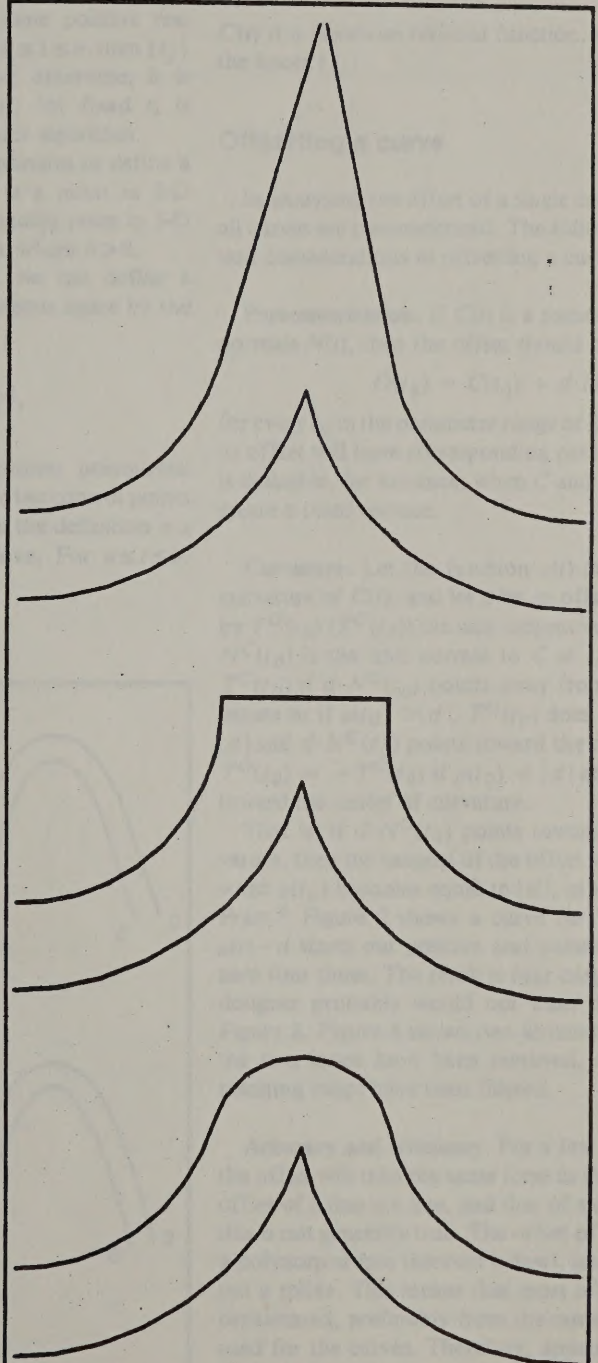


Figure 3. A profile with three possible offsets, all using the same offset distance.

where

- the Q_i are 2-D points called control points;
- t is the parameter, $a \le t \le b$; a and b are fixed, with $0 \le a < b$;
- the $B_{i,k}(t)$ are scalar-valued piecewise polynomials in the variable t, of order k (degree $k-1$).

The $B_{i,k}(t)$ are called the basis functions. They are completely defined by the order k and a knot vector $\{t_j\}_{j=1}^{n+k}$, where

$$\begin{aligned} a = t_1 = t_2 = \ldots = t_k < t_{k+1} \le t_{k+2} \le \ldots \le t_n < t_{n+1} \\ = \ldots = t_{n+k} = b \end{aligned}$$

(nonperiodic basis). If there exists some positive real number r, such that $t_{l+1} - t_l = r$ for all $k \le l \le n$, then $\{t_j\}$ is said to be a uniform knot vector; otherwise, it is nonuniform. The value of a $B_{i,k}(t)$, for fixed t, is calculated with the recursive Cox-deBoor algorithm.

We make use of homogeneous coordinates to define a rational B-spline curve. If $Q = (x,y)$ is a point in 2-D Euclidean space, we denote a corresponding point in 3-D homogeneous space by $Q^h = (hx, hy, h)$, where $h > 0$.

By analogy to Equation 1 above, we can define a polynomial B-spline curve in homogeneous space by the vector equation

$$C^h(t) = \sum_{i=1}^{n} B_{i,k}(t) Q_i^h,$$

where the $B_{i,k}(t)$ are the usual kth-order polynomial B-spline basis functions, and the Q_i^h are the control points in homogeneous space. Also implicit in the definition is a knot vector $\{t_j\}_{j=1}^{n+k}$, as defined above. For $a \le t \le b$, $C^h(t)$ forms a set of points in 3-D homogeneous space. $C(t)$ denotes the 2-D curve associated with $C^h(t)$—i.e., the set of 2-D points formed by the projection of $C^h(t)$ into 2-D space and obtained by dividing the first two coordinates of each point by its third coordinate. $C(t)$ is called the rational B-spline curve and is defined by

$$C(t) = \frac{\sum_{i=1}^{n} B_{i,k}(t) h_i Q_i}{\sum_{i=1}^{n} B_{i,k}(t) h_i}$$

$C(t)$ is a piecewise rational function, with breakpoints at the knots $\{t_j\}$.

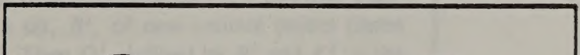


Figure 4. Curve and offset: with loops removed (a); with resulting cusps filleted (b).

Offsetting a curve

In analyzing the offset of a single curve, we assume that all curves are parameterized. The following are the important considerations in offsetting a curve:

Parameterization. If $C(t)$ is a parameterized curve with normals $N(t)$, then the offset should have the form

$$O(t_0) = C(t_0) + d \cdot N(t_0)$$

for every t_0 in the parameter range of C; i.e., the curve and its offset will have corresponding parameterizations. This is desirable, for instance, when C and O are being used to create a ruled surface.

Curvature. Let the function $\rho(t)$ denote the radius of curvature of $C(t)$, and let d be an offset distance. Denote by $T^O(t_0)$ $(T^C(t_0))$ the unit tangent vector to O (C) at t_0. $N^C(t_0)$ is the unit normal to C at t_0. Then $T^O(t_0) = T^C(t_0)$ if $d \cdot N^C(t_0)$ points away from the center of curvature or if $\rho(t_0) > |d|$. $T^O(t_0)$ does not exist if $\rho(t_0) = |d|$ and $d \cdot N^C(t_0)$ points toward the center of curvature. $T^O(t_0) = -T^C(t_0)$ if $\rho(t_0) < |d|$ and $d \cdot N^C(t_0)$ points toward the center of curvature.

That is, if $d \cdot N^C(t_0)$ points toward the center of curvature, then the tangent of the offset will reverse direction when $\rho(t_0)$ becomes equal to $|d|$, as shown by Faux and Pratt.[6] Figure 2 shows a curve for which the function $\rho(t) - d$ starts out positive and passes through the value zero four times. The result is four cusps and two loops. A designer probably would not want the offset shown in Figure 2. Figure 4 shows two alternative solutions: in (a) the two loops have been removed, and in (b) the two resulting cusps have been filleted.

Accuracy and efficiency. For a few simple curve types, the offset will take the same form as the curve; that is, the offset of a line is a line, and that of a circle is a circle. But this is not generally true. The offset of a polynomial is not a polynomial (see theorem below), and that of a spline is not a spline. This means that most offsets have to be approximated, preferably from the same space of functions used for the curves. Therefore, approximation accuracy, numerical stability, and efficiency in terms of computational speed and required amount of data become important issues.

Theorem. Let $C(t) = (x(t), y(t))$ be a curve. Assume that $x(t)$ and $y(t)$ are polynomials. Let $O(t) = (x_O(t), y_O(t))$ be the offset of $C(t)$, and assume that $x_O(t)$ and $y_O(t)$ are polynomials. Then $C(t)$ is a straight line.

Proof. Let $(\bar{x}(t), \bar{y}(t))$ denote the first derivative vector of $C(t)$. For brevity we drop t and use the notation x, y, x_O, y_O, $\bar{x}$, $\bar{y}$. Since x_O and y_O are polynomials,

$$x - d(\bar{y}/\sqrt{\bar{x}^2 + \bar{y}^2}) \text{ and } y + d(\bar{x}/\sqrt{\bar{x}^2 + \bar{y}^2})$$

are polynomials. It follows that

$$p = \bar{x}/\sqrt{\bar{x}^2 + \bar{y}^2} \text{ and } q = \bar{y}/\sqrt{\bar{x}^2 + \bar{y}^2}$$

are also polynomials and that $p^2 + q^2 = 1$ for all t. But the conditions $p^2 + q^2 = 1$, $p^2 \geq 0$, and $q^2 \geq 0$ imply that p^2 and q^2 are both bounded. Now the only bounded polynomials are the constants, and hence p and q are constants. Since (p,q) is the tangent vector of $C(t)$, it follows that $C(t)$ is a straight line.

For the remainder of this article, we assume that the underlying curve and offset representation is nonuniform rational B-splines.

Methods. We have implemented and tested extensively two methods of creating offsets of curves: (1) subdivision and offsetting of the control net and (2) fitting techniques.

Subdivision. Subdivision was suggested to us by Blomgren,[7] who first implemented it at Boeing. The algorithm contains these steps:

(1) Let C be the curve and d the offset distance. Denote by A^1 the set of original control points of C, and by K^1 the corresponding knot vector. Set $i = 1$.

(2) Offset each leg of A^i (as line segments) a distance d.

(3) Intersect the offset polygon legs in a pairwise manner. This yields a set, B^i, of new control points (same number as in A^i). Then O^i, defined by B^i and K^i, is the ith approximation to the offset of C.

(4) Check the deviation of O^i from the true offset.

(5) If the deviation is greater than or equal to epsilon, subdivide A^i and K^i to get A^{i+1} and K^{i+1} (see Cohen, Lyche, and Riesenfeld[8]). Then set $i \leftarrow i+1$, and go back to step 2. If the deviation is less than epsilon, stop.

The implementation of step 4 requires careful thought. Modeling applications, required accuracy and performance, degree of the B-splines—all play a role in determining what type of norm and how many sample points in each span should be used for the convergence check. For illustration, we describe the implementation of step 4 using a uniform norm with one sample point from each span:

Let t_m denote the middle value of the jth span of K^i. Let $Q = O^i(t_m)$. Project Q onto C along the normal to O^i at t_m; denote the projection by R. If $||Q-R| - |d|| \leq$ epsilon, then we have convergence in the jth span. In this case, go to the next span. If $||Q-R| - |d|| >$ epsilon, then add t_m to the new knot vector K^{i+1} and go to the next span.

Repeat the procedure for each span. Subsequently, if any values have been added to K^{i+1}, then subdivide; if $K^i = K^{i+1}$, stop.

Figure 5 shows three typical B-spline curves and their offsets, and Table 1 summarizes the covergence data we obtained by offsetting these curves with the algorithm described above. Epsilons of one percent and 0.1 percent of the offset distance were used. The offsets shown in Figure 5 are the results with epsilon set to one percent. In the table, "order" refers to the order of the B-spline basis functions, Nl is the number of control points defining the original curve, N2 is the number of control points used to represent the offset, and IC is the number of iterations required for convergence.

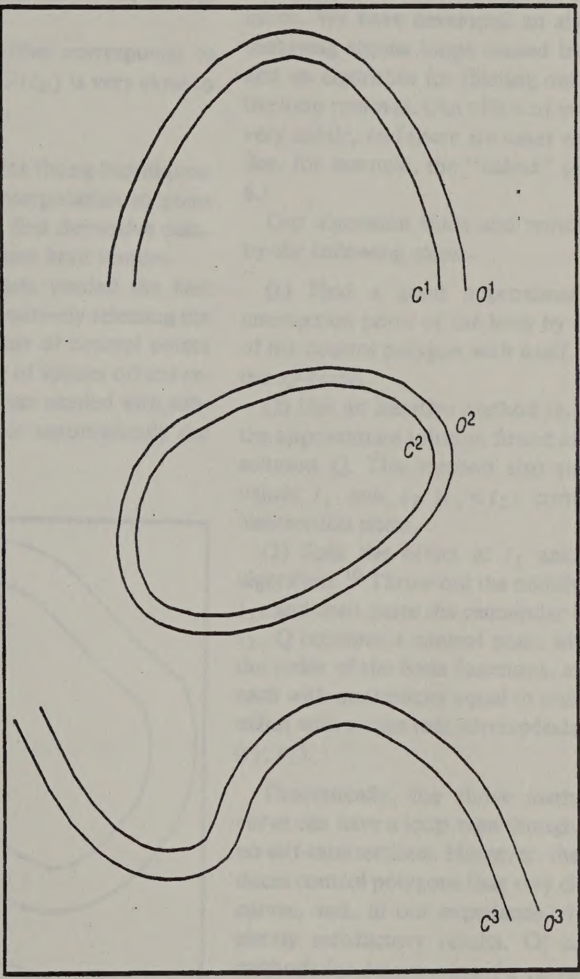


Figure 5. Three splines with offsets. See offset data in Table 1.

Table 1.
Offset and convergence data for the three splines in Figure 5.

SPLINE	EPSILON	ORDER	N1	N2	IC
C1	1%	4	5	13	4
C1	0.1%	4	5	35	6
C2	1%	3	9	19	3
C2	0.1%	3	9	35	5
C3	1%	4	12	21	2
C3	0.1%	4	12	49	5

There are several advantages of subdivision:

- It is elegant and completely general. All polynomial and rational B-spline curves are handled with one algorithm. If the control points are in homogeneous representation (rational curves), then the homogeneous weights (third coordinates) are simply carried over from the set A^i to B^i (step 3 above).
- Lines and circles are offset precisely (not approximated) with one iteration.
- Building in a convergence check is easy.
- Finding and recording the positions of cusps and loops during the offsetting process are easy because the offset changes direction if and only if the control net changes direction.
- The parameterization of the offset corresponds to that of the original curve; i.e., $O(t_0)$ is very close to $C(t_0) + d \cdot N(t_0)$ for every t_0.

Fitting. We have tried three different fitting techniques: least squares fitting to point data, interpolation to point data, and interpolation to point and first derivative data. We used both uniform and nonuniform knot vectors.

Of the three methods, least squares yielded the best results on the most examples. By interactively selecting the number of data points and the number of control points for the fit, we produced for a number of splines offsets requiring slightly fewer control points than needed with subdivision. But it is difficult to estimate automatically the number of data points and control points needed. This fact, together with the advantages of subdivision, led us t abandon fitting techniques.

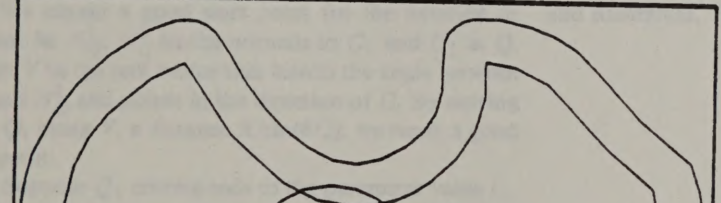


Figure 6. Island problem: offset intersects itself, resulting in two islands.

Before leaving this section, we point out that Klass[9] has used another method, basically a subdivision technique, to offset cubic splines. He determines a cubic expression for the offset in terms of the tangent vectors and curvatures at the end points of the curve segment and its offset.

Loop removal

Figure 2 shows an example of an offset with two simple loops. We have developed an algorithm for finding and removing simple loops caused by the condition $\rho < |d|$ and an algorithm for filleting out the cusp resulting from the loop removal. (An offset of even a simple curve can be very exotic, and there are cases we will not consider here. See, for example, the "island" problem shown in Figure 6.)

Our algorithm finds and removes loops automatically by the following steps:

(1) Find a good approximation, Q', to the self-intersection point of the loop by finding the intersections of the control polygon with itself, a very fast operation in the *xy*-plane.

(2) Use an iterative method (e.g., Newton) to improve the approximate solution found above. Call the improved solution Q. This method also yields the two parameter values t_1 and t_2 $(t_1 < t_2)$ corresponding to the self-intersection point.

(3) Split the offset at t_1 and t_2, using the Boehm algorithm.[10] Throw out the middle section between t_1 and t_2, and then paste the remainder back together at t_1 and t_2. Q becomes a control point with multiplicity equal to the order of the basis functions, and t_1 and t_2 are knots, each with multiplicity equal to order-1. This step yields an offset with a cusp that corresponds to the parameter range (t_1, t_2).

Theoretically, the above method can fail because a curve can have a loop even though its control polygon has no self-intersections. However, the offsetting process produces control polygons that very closely approximate their curves, and, in our experience, this method yields completely satisfactory results. Of course, there are other methods for determining if a curve has a loop, such as the use of the rotation index, as described by Do Carmo.[11]

Figure 7a shows a flat surface between a curve and its offset. The offset had one loop, which was removed. In Figure 7b the cusp resulting from loop removal was filleted. The correspondence between the parameterizations can be clearly seen. The filleting method, illustrated in Figure 8, is as follows:

(1) Specify a fillet radius, R.

(2) The offset is split at the cusp, Q, resulting in the two curves, C_1 and C_2.

(3) Determine the center point C of the circle with radius R, which touches and is tangent to C_1 and C_2 at the points Q_1 and Q_2. The center C will be at the intersection of O_1 and O_2, which are the offsets of C_1 and C_2 at a distance R. Since parameterizations and tangents correspond be-

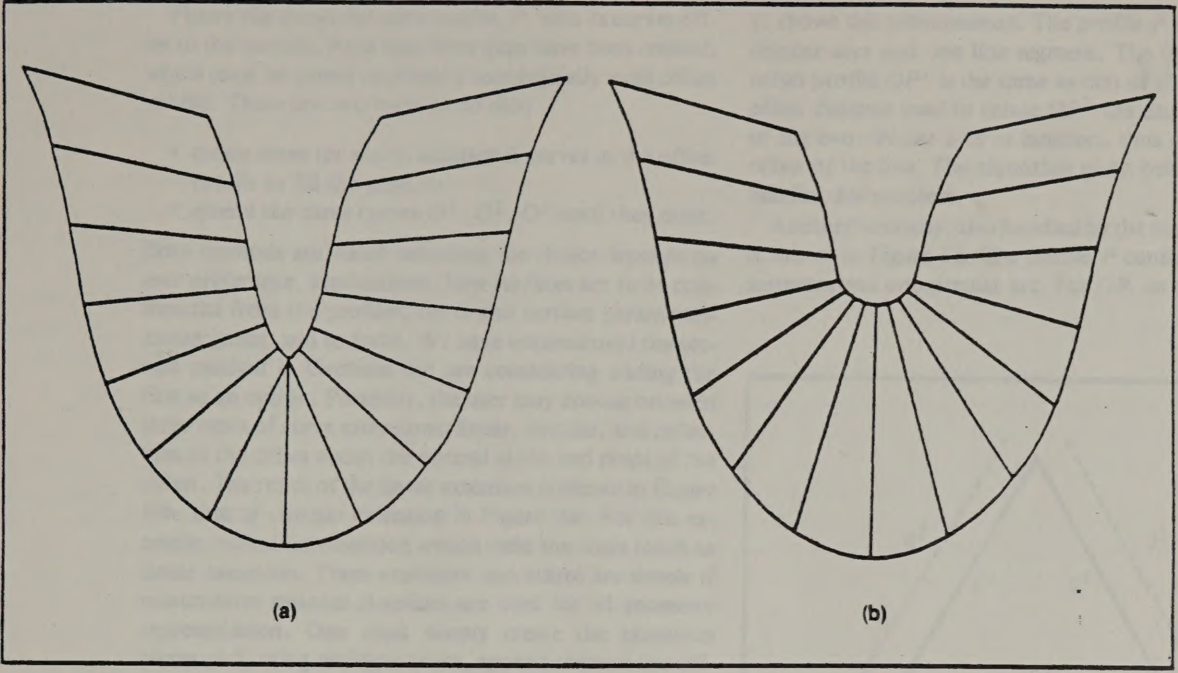


Figure 7. Flat surface between curve and offset: with cusp (a); with cusp filleted (b).

tween the curves and their offsets, we can compute points and tangents on the offsets without generating explicit representations of them. Hence, we can use a Newton method to intersect O_1 and O_2, with C_1, C_2, and R as input. We obtain a good start point for the iteration as follows: let N_Q^1, N_Q^2 be the normals to C_1 and C_2 at Q, and let V be the unit vector that bisects the angle between N_Q^1 and N_Q^2 and points in the direction of C. By moving from Q, along V, a distance $R/\sin(\theta/2)$, we reach a good start point.

(4) Suppose Q_1 corresponds to the parameter value t_1, and Q_2 to t_2. Let t_s be the starting parameter value of C_1 and let t_e be the ending parameter value of C_2. Split C_1 at t_1 to get the curve C_1' defined on (t_s, t_1). Split C_2 at t_2 to get the curve C_2' defined on (t_2, t_e). Create the rational B-spline representation[2] of the circular arc from Q_1 to Q_2 with radius R and center at C. Parameterize this arc to the range (t_1, t_2). Combine C_1', C_2' and the arc into a composite rational B-spline curve with multiple knots at t_1 and t_2.

Offsetting a 2-D profile

In this section, we address the problem of offsetting a profile of the form $P = \{C^i\}_{i=1}^{n}$. Starting with two simple but basic examples and progressing to some more complex cases, we conclude with a general algorithm, which handles most of the offset problems likely to be encountered in practice.

Figure 9a shows a closed profile consisting of three curves (lines) and their offsets. The offsets are to the inside of the profile. The required offset profile, denoted by OP in Figure 9b, is obviously obtained by clipping back (intersecting) the three offsets O^1, O^2, O^3. A good routine, capable of finding all curve-curve intersection points, is essential for offsetting profiles. We use a subdivision technique based on the algorithm given by Lane and Riesenfeld.[12]

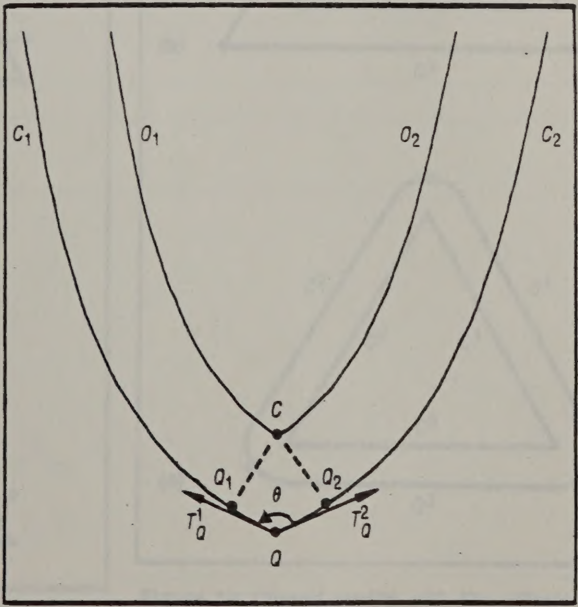


Figure 8. Filleting out a cusp in an offset.

Figure 10a shows the same profile, *P*, with its curves offset to the outside. Note that three gaps have been created, which must be closed to obtain a topologically valid offset profile. There are two ways to do this:

- create three (or more) additional curves in the offset profile to fill the gaps, or
- extend the three curves O^1, O^2, O^3 until they meet.

Both methods are viable solutions; the choice depends on user preference, applications, how surfaces are to be constructed from the profiles, curve and surface parameterization issues, and so forth. We have implemented the second method in Geomod but are considering adding the first as an option. Presently, the user may choose between three types of curve extensions: linear, circular, and reflection of the offset about the normal at the end point of the offset. The result of the linear extension is shown in Figure 10b, that of circular extension in Figure 10c. For this example, reflection extension would yield the same result as linear extension. These extension operations are simple if nonuniform rational B-splines are used for all geometry representation. One must simply create the extension pieces and, using multiple knots, append them to the offsets.

Another topological anomaly which can easily occur in offsetting a profile is the disappearance of a curve. Figure 11 shows this phenomenon. The profile *P* consists of two circular arcs and one line segment. The topology of the offset profile OP^1 is the same as that of *P*, but the larger offset distance used to create OP^2 has caused the offsets of the two circular arcs to intersect, thus eliminating the offset of the line. The algorithm given below detects and handles this problem.

Another anomaly, also handled by the algorithm below, is shown in Figure 12. The profile *P* consists of two line segments and one circular arc. For *OP*, an offset distance

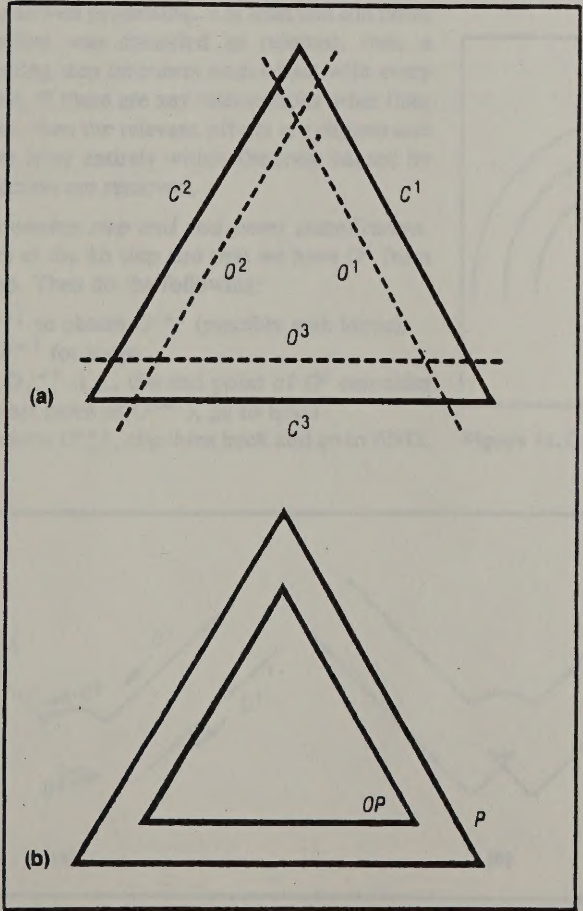


Figure 9. Closed profile with the offsets to its curves (offsets to the inside) (a); offset profile (b).

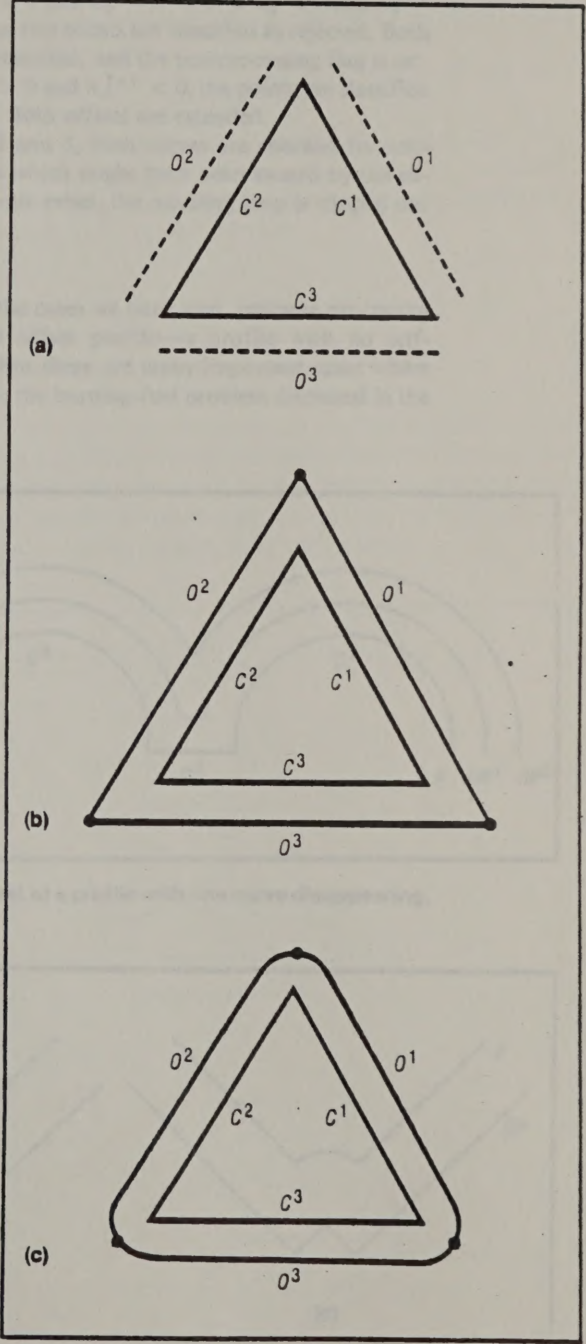


Figure 10. Closed profile with the offsets to its curves (offsets to the outside) (a); offset profile using linear extensions (b); offset profile using circular extensions (c).

greater than the radius of the arc was used; as a result, the offset of the arc changed direction. When the offsets are extended (linearly) to meet, the extensions to the arc offset cross each other (Figure 12b). The loop removal routine is applied, yielding the result shown in Figure 12c.

General algorithm. As the algorithm for offsetting a profile is complicated, we present it in several steps, each successive step containing a bit more detail.

Let $P = \{C^i\}_{i=1}^{n}$ be a profile. We assume that $n > 1$. Denote the offset of P by OP. The algorithm is roughly as follows:

Process the curves of P in a pairwise fashion; i.e.,

$$(C^1, C^2), (C^2, C^3), \ldots, (C^{n-1}, C^n).$$

At each step, offset the two curves, process any loops and/or cusps that are created, and ensure that the offsets join properly. This last step usually involves either clipping back or extending the two offsets.

This simple algorithm would be entirely sufficient if not for the unpleasant special cases, such as those shown in Figures 11 and 12. For such cases, the following logic is built into the algorithm:

- At each step of the pairwise processing, certain conditions are checked and the end points of each offset are classified as being either accepted or rejected. If both end points of an offset O are classified as rejected, then O is immediately rejected (during the pairwise processing); i.e., it does not become a part of OP.
- After the pairwise processing, if at least one end point of one offset was classified as rejected, then a postprocessing step intersects each offset with every other offset. If there are any intersections other than at the joins, then the relevant offsets are clipped and any offsets lying entirely within the loop caused by the intersection are removed.

Pairwise processing step and end point classification. Suppose we are at the ith step and that we have O^i from the $(i-1)$th step. Then do the following:

- Offset C^{i+1} to obtain O^{i+1} (possibly with loops).
- Process O^{i+1} for loops.
- If $O_e^i = O_s^{i+1}$ (i.e., the end point of O^i coincides with the start point of O^{i+1}), go to END.
- If O^i intersects O^{i+1}, clip them back and go to END.
- Do end point classification (and possibly offset extension) as follows:

  Calculate the tangent T_e^i to O^i at O_e^i and the tangent T_s^{i+1} to O^{i+1} at O_s^{i+1}. Form the lines $L^i = O_e^i + \lambda^i \cdot T_e^i$ and $L^{i+1} = O_s^{i+1} + \lambda^{i+1} \cdot T_s^{i+1}$. If the lines intersect, denote by λ_o^i and λ_o^{i+1} the parameter values corresponding to the point of intersection. We distinguish three cases:

  (1) If L^i and L^{i+1} do not intersect, or if $\lambda_o^i < 0$ and $\lambda_o^{i+1} > 0$, then O_e^i and O_s^{i+1} are classified as rejected. The offsets are not extended, and the postprocessing flag is set.

  (2) If $\lambda_o^i > 0$ and $\lambda_o^{i+1} > 0$, or if $\lambda_o^i < 0$ and $\lambda_o^{i+1} < 0$, then the two points are classified as rejected. Both offsets are extended, and the postprocessing flag is set.

  (3) If $\lambda_o^i > 0$ and $\lambda_o^{i+1} < 0$, the points are classified as accepted. Both offsets are extended.

  In cases 2 and 3, both curves are checked for self-intersections which might have been caused by the extensions. If one exists, the resulting loop is clipped out of the curve.
- END

In most of the cases we have seen, pairwise processing yields a valid offset profile—a profile with no self-intersections. But there are many important cases where this is not true: the burning-fuel problem discussed in the

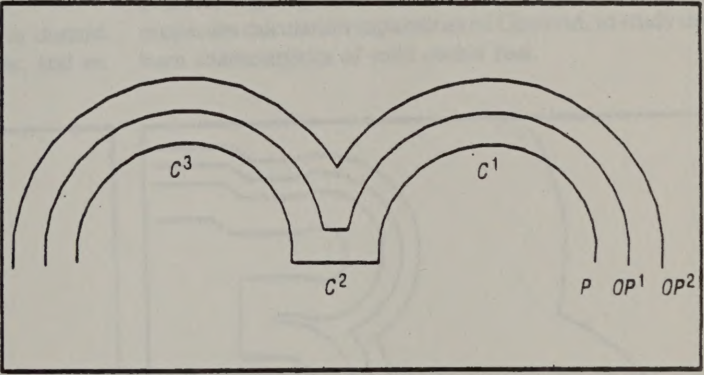


Figure 11. Offset of a profile with one curve disappearing.

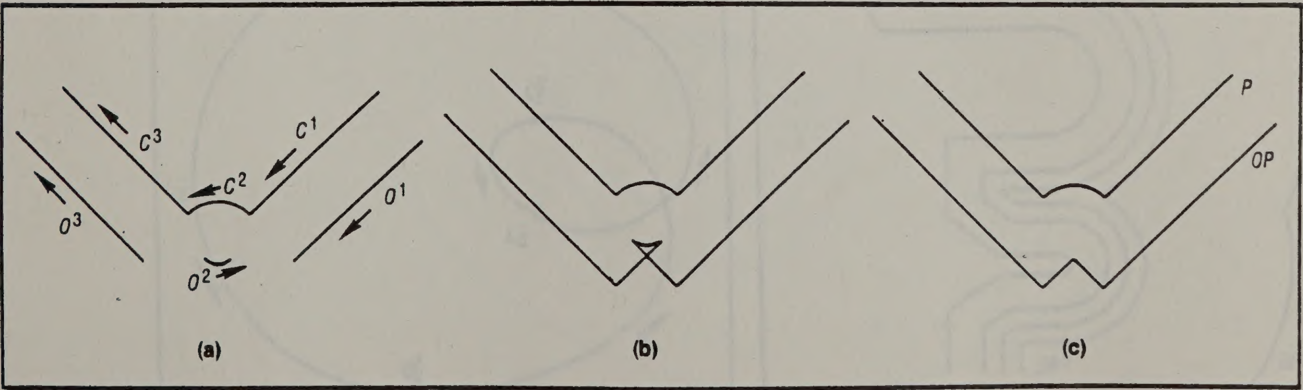


Figure 12. Profile, together with offsets to its curves: arc has switched directions (a); extensions to the offset arc intersect (b); final offset to profile (c).

next section, for example. Also, depending on the offset distance, the simple example shown in Figure 11 may require postprocessing.

The offset profiles resulting from the pairwise processing can possess multiple self-intersections, which form nested and/or non-nested loops. The postprocessing problem, then, is basically one of loop removal. There are two aspects to loop removal (for profiles or simple curves):

- finding the intersection points, and
- deciding which portions of the geometry to discard when an intersection is found.

In the section on loop removal above, we gave an algorithm for finding the self-intersection points of a single B-spline curve. The corresponding problem for profiles is to find all the curve-curve intersection points. Subdivision, as described by Lane and Riesenfeld,[12] is a reliable global method.

For single offsets, we simply traversed the offset and threw away the loops as we found them. This is usually adequate for open geometry, but closed geometry is trickier. Consider the closed profile shown in Figure 13. Consisting of three curves, the profile is oriented in a counterclockwise direction, and the start points of the curves are shown. It contains three loops: two small ones, L_1 and L_3, and a larger one, L_2, formed by clipping L_1 and L_3 out of the profile. Since its start point is in L_1, if we applied the simple loop removal logic described above to this profile, we would be left with L_1. But, intuitively, the answer we want is L_2.

Any algorithm that decides what geometry to discard from a closed profile with loops will be heuristic, and we do not intend to treat this problem in detail here. Such an algorithm must have a set of criteria on which to base its decisions, however, and one or more of the following methods might be used:

(1) Carry the curve end point classifications assigned during pairwise processing into the postprocessing step. Consider (in some way) the number of "accepted" versus "rejected" classifications in each loop; for instance, keep the loop in which the least number of "rejected" classifications occurred.

(2) Consider the number of curves in each loop; keep the loop with the most curves.

(3) Consider the relative arc lengths of the loops; keep the longest loop.

(4) Consider loop containment (nesting); discard any loops contained in another loop.

We use a combination of methods 1 and 2 with very satisfactory results. In our experience, most loops encountered in offsetting are of type L_3 (nested); method 4 is entirely sufficient for this type.

A practical application

We have mentioned the use of offset profiles in mold design. Another important application is the use of offset profiles, together with the Boolean operation and mass properties calculation capabilities of Geomod, to study the burn characteristics of solid rocket fuel.

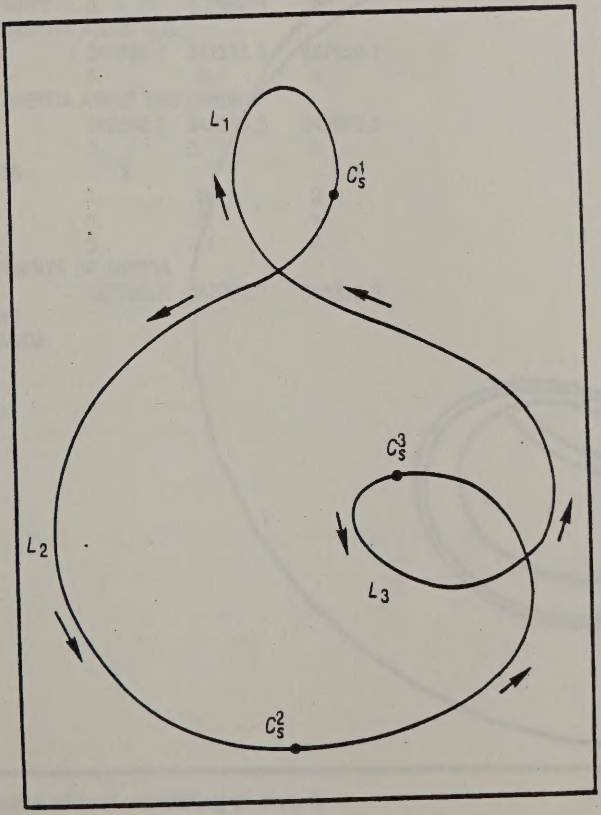


Figure 13. Closed offset profile before loop removal.

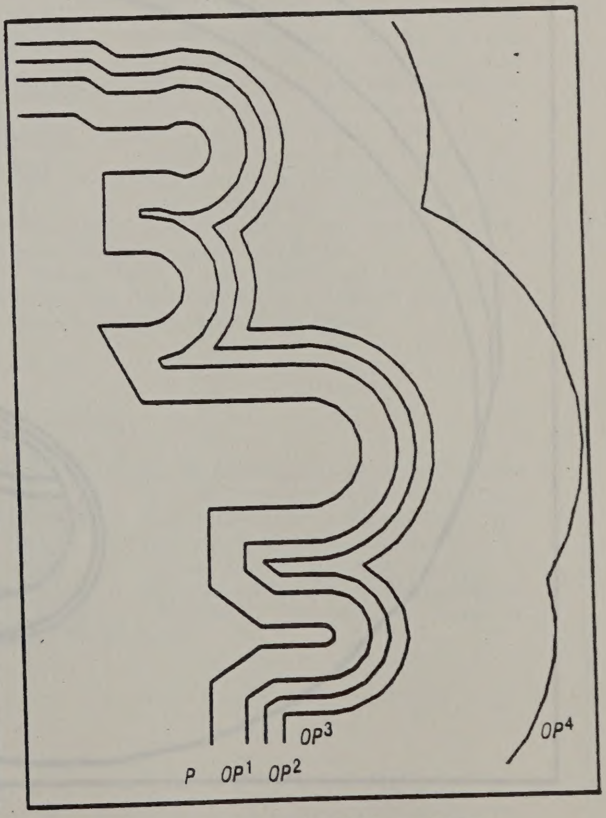


Figure 14. Initial void profile with four offsets.

Consider a casing containing a volume of solid rocket fuel. In the middle of the volume is a void, whose surface represents the initial burn area of the fuel. The fuel burns outward toward the wall of the casing, and, at each time step, the burn area of the fuel can be represented as an offset surface to the initial burn surface. If we assume that casing, fuel, and initial void are all surfaces (solids) of revolution, then we can represent the offset surfaces at each time step by forming the offset profiles and revolving them.

The shape of the initial burn surface is critical in rocket performance, and one design goal is to minimize the variation of the burn surface area as the fuel burns outward. This requires a very convoluted shape for the initial void profile. Profile *P* of Figure 14 is an example. The offsets, OP^1 through OP^4, represent the burn surface as the fuel burns outward. The casing profile is not shown in this figure, but it would be to the right of OP^4.

At each step, the current profile is revolved and its surface area computed. The areas are: 374.4, 448.0, 450.3, 422.9, and 598.7. Figure 15 shows the surface generated by OP^3.

Figure 16 shows the volume of fuel remaining at step 3. It was obtained by cutting the void volume of Figure 15 out of the casing volume. Also shown are the mass properties of the fuel volume.

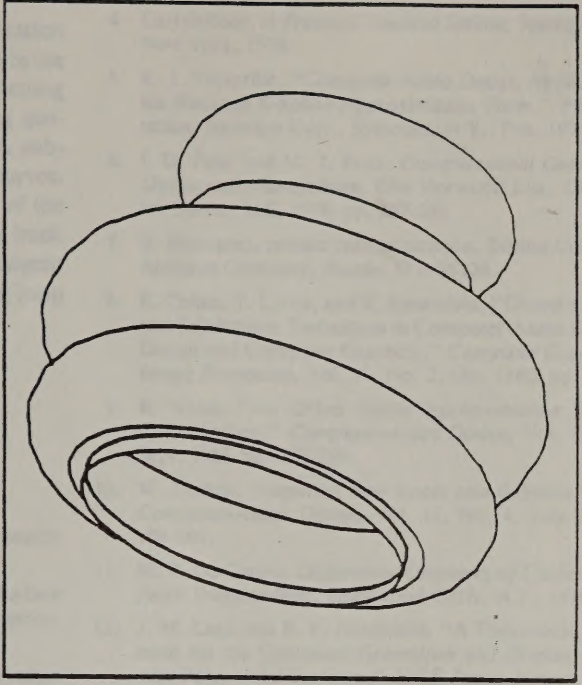
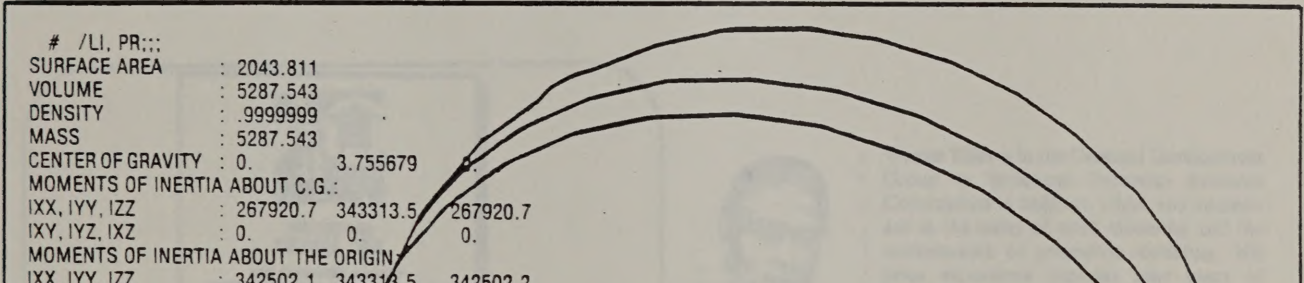


Figure 15. The surface generated by the offset OP^3 of Figure 14.



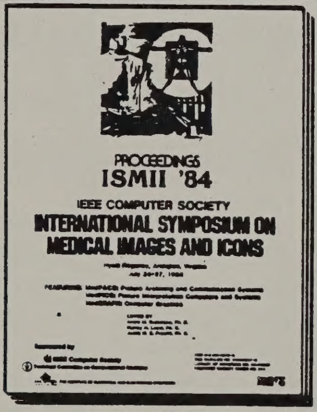
```
# /LI, PR;;;
SURFACE AREA        : 2043.811
VOLUME              : 5287.543
DENSITY             : .9999999
MASS                : 5287.543
CENTER OF GRAVITY   : 0.          3.755679     0.
MOMENTS OF INERTIA ABOUT C.G.:
IXX, IYY, IZZ       : 267920.7    343313.5     267920.7
IXY, IYZ, IXZ       : 0.          0.           0.
MOMENTS OF INERTIA ABOUT THE ORIGIN:
IXX, IYY, IZZ       : 342502.1    343313.5     342502.2
IXY, IYZ, IXZ       : 0.          0.           0.
PRINCIPAL AXES             X           Y            Z
1                   : 1.          0.           0.
2                   : 0.          0.           1.
3                   : 0.          -1.          0.
PRINCIPAL MOMENTS OF INERTIA
I11, I22, I33       : 267920.7    267920.7     343313.5
LIST__CURRENT
ENTER COMMAND#
```

Figure 16. Volume of fuel remaining at step 3.

We have presented an overview of the implementation of an offset capability for planar curves and profiles in the solid modeler Geomod. In the sections on curve offsetting and loop removal, we assumed that the underlying geometric form was the nonuniform rational B-spline. A subdivision algorithm was presented for offsetting curves. The section on profile offsetting was independent of the form of geometry representation, relying only on a basic set of geometric utilities, including point and tangent calculation, curve offsetting, intersection, extension, loop removal, and filleting. ■

References

1. *GEOMOD 2.5 User Manual*, Structural Dynamics Research Corp., 2000 Eastman Dr., Milford, OH 45150.
2. Wayne Tiller, "Rational B-Splines for Curve and Surface Representation," *IEEE Computer Graphics and Applications*, Vol. 3, No. 6, Sept. 1983, pp. 61-69.
3. R. F. Riesenfeld, "Applications of B-Spline Approximation to Geometric Problems of Computer-Aided Design," PhD dissertation, Syracuse Univ., Syracuse, N.Y., May 1973.
4. Carl deBoor, *A Practical Guide to Splines*, Springer-Verlag, New York, 1978.
5. K. J. Versprille, "Computer-Aided Design Applications of the Rational B-Spline Approximation Form," PhD dissertation, Syracuse Univ., Syracuse, N.Y., Feb. 1975.
6. I. D. Faux and M. J. Pratt, *Computational Geometry for Design and Manufacture*, Ellis Horwood Ltd., Chichester, W. Sussex, UK, 1979, pp. 267-268.
7. B. Blomgren, private communication, Boeing Commercial Airplane Company, Seattle, WA 98124.
8. E. Cohen, T. Lyche, and R. Riesenfeld, "Discrete B-Splines and Subdivision Techniques in Computer-Aided Geometric Design and Computer Graphics," *Computer Graphics and Image Processing*, Vol. 14, No. 2, Oct. 1980, pp. 87-111.
9. R. Klass, "An Offset Spline Approximation for Plane Cubic Splines," *Computer-Aided Design*, Vol. 15, No. 5, Sept. 1983, pp. 297-299.
10. W. Boehm, "Inserting New Knots into B-Spline Curves," *Computer-Aided Design*, Vol. 12, No. 4, July 1980, pp. 199-201.
11. M. P. Do Carmo, *Differential Geometry of Curves and Surfaces*, Prentice-Hall, Englewood Cliffs, N.J., 1976.
12. J. M. Lane and R. F. Riesenfeld, "A Theoretical Development for the Computer Generation and Display of Piecewise Polynomial Surfaces," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. PAMI-2, No. 1, Jan. 1980, pp. 35-46.



PROCEEDINGS
ISMII '84
IEEE COMPUTER SOCIETY
INTERNATIONAL SYMPOSIUM ON MEDICAL IMAGES AND ICONS

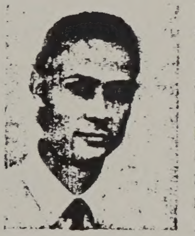
Medical Symposium proceedings features MedPACS: picture archiving and communication systems, MedPICS: picture interpretation computers and systems and MedGRAPH: computer graphics.

Order #544

Proceedings—International Symposium on Medical Images and Icons-July 1984

Nonmembers—$54.00
Members—$27.00
Handling Charges Extra

Order from IEEE Computer Society Order Dept.
PO Box 80452, Worldway Postal Center
Los Angeles, CA 90080 USA
(714) 821-8380



Wayne Tiller is in the Geomod Development Group at Structural Dynamics Research Corporation in Milford, Ohio. His interests are in the areas of solid modeling and the mathematics of geometric modeling. His prior experience includes nine years of mathematics and computer work in Germany, including positions with the Burroughs Corporation and the Army and Air Force University in Munich and the Computer Center for the Sciences in West Berlin.

Tiller received his BS in mathematics from Louisiana State University at Baton Rouge and his MS and PhD in mathematics from Texas Christian University in Fort Worth.

Eric G. Hanson is a scientific programmer for SDRC, where he works on the Geomod solid modeling system. His interests include graphical algorithms and engineering workstations. Hanson received a BS degree in quantitative analysis from the University of Cincinnati in 1972. He is a member of ACM.

The authors' address is Structural Dynamics Research Corp., 2000 Eastman Dr., Milford, OH 45150.

APPENDIX F

Table of Contents
Appendix F

F.1 Common Blocks.. 1

F.1.1 ASCII Codes.. 1

F.1.2 Channel Numbers.. 1

F.1.3 Operational Parameters.. 2

F.1.4 File Name Extensions.. 5

F.2 Suggested Variable Names.. 6

F.3 Subroutine Descriptions.. 7

F.3.1 ARC Intersection.. 7
- MBRCHK - Check MBR's for overlap.. 7
- CROSS - Compute geometric relationship between two line segments.. 8
- HORINT - Compute intersections with horizontal line.. 9
- VERINT - Compute intersections with vertical lines.. 10

F.3.2 Zone Generation.. 11
- VZONE - Generate offset line from edge.. 11
- LOOP - Remove loops in edge.. 12
- ZCURVE - Cusp resolution.. 13

F.3.3 Polygon to ARC.. 14

F.3.4 Feature Verification/Formation.. 15

F.3.5 DBMS Shell.. 17
- DBOPN - Open attributes database.. 17
- CHKKEY - Validate field name.. 18
- DBGROW - Read row from table.. 19
- DBPROW - Store row in table.. 21
- DBGVAR - Get field value from row.. 23
- DBPVAR - Store field value in row.. 24
- DBRDEL - Delete a row from table.. 25
- DBSEAR - Search attributes for occurrence.. 26
- DBSLST - List contents of logical schema.. 28
- DUMPAT - Generic internal to ASCII Dump.. 29
- RELCOM - Load logical schema.. 30

F.3.7 Spatial Database Primitives.. 32
- OPNMAP - Open arc/node map.. 32
- DELMAP - Delete MOSS map 34
- CREMAP - Create MOSS arc/node map.. 35
- RENAME - Rename MOSS map.. 36
- CRENOD - Create node record.. 37
- UPNOD - Update node record.. 38

DELNOD - Delete node.. 39
LSTNOD - List contents node record.................. 40
FNDNOD - Find node.. 41
ADDARC - Create edge record............................ 42
DELARC - Delete edge record............................ 43
FNDARC - Find edge record............................... 44
GSTARC - List contents arc record.................... 45
UPARC - Update arc record.............................. 46
ADDCOR - Add coordinate record....................... 47
DELFET - Delete a feature................................ 48
BLDFET - Build a feature using chain file........... 49

F. System Specifications

This section describes the necessary common blocks, suggested variable names, and subroutine descriptions. The subroutine descriptors are related to the major modules identified in section 3.0.

F.1 Common blocks:

The following common blocks are suggested:

F.1.1 NAME CHARS

```
COMMON /CHARS /ICHARS(128)
```

Description :
This common block contains the ASCII codes for the first 128 characters in the ASCII table.

Variable/Array Definition: ICHAR is a 128 word array. Suggested type is Integer*2.

F.1.2 NAME CHANL

```
COMMON /CHANL /IO(64)
```

Description:
This common contains a list of channel numbers for disk IO, terminal IO, and graphics peripheral IO.

Variable/Array Definition

WORD	CONTENTS	# BYTES	TYPE	DESCRIPTION
1	1	2	Int*2	MASTER.DT file
2	2	2	Int*2	WORK.DT file

3	3	2	Int*2	Free
4	4	2	Int*2	Free
5	5	2	Int*2	Free
6	6	2	Int*2	Free
7	7	2	Int*2	Free
8	8	2	Int*2	Graphics Input
9	9	2	Int*2	Graphics Output
10	10	2	Int*2	Terminal Output
11	11	2	Int*2	Terminal Input
12	12	2	Int*2	Line Printer
13	13	2	Int*2	Free
14	14	2	Int*2	Free
15	15	2	Int*2	DESCRIBE.FA
16	16	2	Int*2	free
17	17	2	Int*2	INTERPRO
18	18	2	Int*2	COMMAND.DT
19-64				Free

F.1.3 NAME PARAMS

```
COMMON /PARAMS /IPARM(64)
```

Description:

This common block keeps track of all the record lengths for arc/node records, machine limits, error flags, and certain MOSS limits.

Variable/Array Definition *

WORD	CONTENTS	# BYTES	TYPE	DESCRIPTION
1	4	4	I	Bytes/word
2	32	4	I	Bits/word

* Assuming 32 bit word

3	256	4	I	# Bytes per record for map header
4	1024	4	I	# Bytes per physical record for Node, Arc, Feature, Feature pointer file, and coordinate file
5	64	4	I	# Bytes per logical record for Node, Arc, and Feature files
6	4	4	I	# Bytes per logical record for feature pointer file
7	32	4	I	# Bytes per logical record for feature/arc index files
8	8	4	I	# Bytes per logical record for 2-D coordinate data, full 32 bit integer format
9	4	4	I	# Bytes per logical record for 2-D coordinate data, local coordinate system
10				Reserved
11				Reserved
12	?	4	I	Logical record length for cartographic file
13				Reserved
14	16	4	I	# Bytes allowed (including suffix) for map name
15	-2.14 billion	4	I	Minimum integer value Also used for delete flags
16	9999999	4	I	Maximum real value
17	2.14 billion	4	I	Maximum Integer value
18	set*	4	I	MAP.DT First
19	set	4	I	MAP.DT Last
20	set	4	I	Number of maps in .DT
21	40	4	I	Maximum # of active maps
22	set	4	I	# Lines per page

* Set means MOSS will initialize this value at runtime.

23	set	4	I	Last command (code only) issued
24	set	4	I	Current error code (if any) for error processing
25-64				Unused at this time

F.1.4 NAME

COMMON /EXTEND /ISUFF(32)

Description:
This common block contains a list of all valid extensions fo
creating file names.

WORD	CONTENTS	TYPE	DESCRIPTION
1	NULL	INT	
2	DT	INT	NAMES FILE
3	DH	INT	INDEX FILE
4	CT	INT	CONTRUCT FILE
5	AT	INT	ATTRIBUTES FILE
6	HD	INT	MAP HEADER FILE
7	CB	INT	DICHOTOMOUS 6
8	CD	INT	DISCRETE 7
9	CC	INT	CONTINUOUS 8
10	DP	INT	DISPLAY
11	CM	INT	CONSTRUCT HEADER FILE
12	CV	INT	COUNT VALUE
13	AD	INT	CELL ATTRIBUTES
14	ND	INT	NODE FILE
15	AC	INT	SEGMENT FILE
16	FT	INT	FEATURE FILE
17	CO	INT	COORDINATE FILE
18	CA	INT	CARTOGRAPHIC FILE
19	FF	INT	MAP FAST FILE
20	FP	INT	FEATURE POINTER FILE

F.2 Suggested Variable Names

The following is a list of suggested variable names and their meaning.

NAME	TYPE	DESCRIPTION
NARCS	I	Number of arcs in arc file
NODES	I	Number of nodes in node file
NFEATS	I	Number of features in feature file
NFPTS	I	Number of entries in feature pointer file
NCORDS	I	Number of coordinates in coordinate file
NFAST	I	Number of entries in fast file
CSCALE	R	Coordinate scale factor (where is the decimal point)
NODEID	I	Node identifier number
ARCID	I	Arc identifier number
FEATID	I	Feature ID number
ATTPT	I	Attribute pointer
XMIN	R/I	Minimum X value
XMAX	R/I	Maximum X value
YMIN	I/R	Minimum Y value
YMAX	I/R	Maximum Y value
ZMIN	I/R	Minimum Z value
ZMAX	I/R	Maximum Z value
MAPNAM(16)	I	Name of map currently being processed
NODESR	I	Start node
NODESP	I	End node
NODEPT	I	Node pointer
ARCPT	I	Arc pointer
FEATPT	I	Feature pointer
XCOR	I/R	X coordinate value
YCOR	I/R	Y coordinate value
ZCOR	I/R	Z coordinate value

F.3 Subroutine Descriptions

F.3.1 Arc Intersection

Subroutines MBRCHK (MBRI, MBRZ, CHECK, IER)

Description:
This routine checks to see if two minimum bounding rectangles overlap.

INPUT/OUTPUT

NAME	TYPE	I/O	WORDS	DESCRIPTION
MBR1	I/R	I	6	MBR of Feature or Arc A
MBR2	I/R	I	6	MBR of Feature or Arc B in form XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX
CHECK	Log	O	1	False if no overlap, True if overlap
IER	I	O	1	Error return 1 = normal See error codes for actual values

Subroutine Cross (SEG, SEG2, XCOR, YCOR, ICROS, ISIDE,IER)

Description:

This routine computers the geometric relationship between lin segments. If they intersect, it returns the type of intersectio and the intersection point.

NAME	I/O	TYPE	LENGTH in words	DESCRIPTION
SEG	I	I/R	4	End points of line segment one in form X1, Y1, X2, Y2
SEG2	I	I/R	4	End points of line segment two in form X1, Y1, X2, Y2
XCOR	O	I/R	1	Returned X coordinate (if exists)
YCOR	O	I/R	1	Returned Y coordinate (if exists)
ICROS	O	I	1	Returned segment relationship 1= no intersection 2= Segment 1 split at x,y 3= Segment 2 split at x,y 4= Segment cross at x,y 5= Segments are colinear 6= Segments meet at path endpo 7= Segment meets at one endpoi
ISIDE	O	I	1	1= Seg 1 LT Seg 2 2= Seg 1 GT Seg 2
IER	O	I	1	Error return 1= normal

Subroutine HORINT (SEG1, SEG2, XCOR, YCOR, CROSS)

Description:

Compute the intersection of a line segment with a horizontal line. This is a special case.

NAME	I/O	TYPE	LENGTH	DESCRIPTION
SEG 1	I	I/R	4	Horizontal Segment
SEG 2	I	I/R	4	Non-horizontal Segment in form X1, Y1, X2, Y2
XCOR	O	I/R	1	Returned X intersection point
YCOR	O	I/R	1	Returned Y intersection point
CROSS	O	I	1	Type of intersection 1= not intersection 2= Segment 1 split at x,y 3= Segment 2 split at x,y 4= Segment cross at x,y 5= Segments are colinear 6=Segments meet at endpoints (congruent) 7= Segments meet at one endpoint

Subroutine VERINT (SEG1, SEG2, XCOR,YCOR,CROSS)

Description:

Compute the intersection of a vertical line segment with a line. This is a special case.

NAME	I/O	TYPE	LENGTH	DESCRIPTION
SEG1	I	I/R	4	Vertical Segment
SEG2	I	I/R	4	Non vertical Segment in form X1,Y1,X2,Y2
XCOR	O	I/R	1	Returned X intersection point
YCOR	O	I/R	1	Returned Y intersection point
CROSS	O	I	1	Type of intersection 1= No intersection 2= Segment 1 split at x,y 3= Segment 2 split at x,y 4= Segment cross at x,y 5= Segments are colinear 6=Segments meet at endpoints (congruent) 7= Segments meet at one endpoint

F.3.2 Generate Zone around ARC

Subroutine VZONE (DIST, XCOR, YCORR, NPTS, XOUT, YOUT, NPTS1, IER)

Description:

Calculate a line parallel but offset from an input arc

NAME	I/O	TYPE	LENGTH	DESCRIPTION
DIST	I	R	1	Distance for line offset
XCOR	I	I/R	NPTS	X coordinates for arc
YCOR	I	I/R	NPTS	Y coordinates for arc
XOUT	O	I/R	NPTS+NPTS	Returned x coordinates for new line
YOUT	O	I/R	NPTS+NPTS	Returned y coordinates for new line
NPTS1	O	I	1	Returned number of coordinate pairs
IER	O	I	1	Error return 1= normal

Subroutine LOOP (XCOR, YCOR, NPTS, IER)

Description:

This routine resolves self crossing loops in lines created by subroutine VZONE.

NAME	I/O	TYPE	LENGTH	DESCRIPTION
XCOR	I	I/R	1(dynamic)	X coordinates for arc
YCOR	I	I/R	1(dynamic)	Y coordinates for arc
NPTS	I	I/R	1	# of coordinate pairs in input arc
NPTS	O	I	1	# coordinate pairs in output arc
IER	O	I	1	error return 1= normal

Subroutine ZCURVE (SEG1, SEG2, XCOR, YCOR, XT, YT, DIST, IDO)

Description:

This routine takes two line segments that meet at a point (there endpoints) and based on the value of IDO creates a new set of coordinates to curve smoothly around a pivot point.

NAME	I/O	TYPE	LENGTH	DESCRIPTION
SEG1	I	I/R	4	Endpoints for segment 1 in form X1, Y1, X2, Y2
SEG2	I	I/R	4	Endpoints for segment 2 in form X1, Y1, X2, Y2
XCOR	O	I/R	set in calling	Returned X coordinates for smoothed junction
YCOR	O	I/R	set in calling	Returned Y coordinates for smoothed junction
XT	I	I/R	1	X coordinate for pivot point
YT	I	I/R	1	Y coordinate for pivot point
DIST	I	R	1	Smoothing Distance from pivot point
IDO	I	I	1	Smoothing option = 0 no action = 1 straight across at DIST = 2 triangulate at DIST = 3 insert smooth curve using 10 points at dist

F.3.3 Polygon to ARC

PROGRAM POLYARC

Description:

This routine takes an existing MOSS map and creates a new arc/node map. This is the control module. It asks for old map name, new map name, creates the header, and creates the arc and node files necessary for performing the arc to polygon process to create a final arc node map.

This routine uses the line intersection module, the arc intersection routine from the overlay processor, and the database primitive module.

F.3.4 Feature Verification

PROGRAM VERFIT

Description:
This is the driver for the arc to polygon verification and build process. It calls a series of routines for performing the verification procedure. These routines in turn call the database primitives and a number of special math functions. Appendix C has a detailed description of the verification procedure.

Subroutine LANGLE (SEG1, SEG2, ANGLE)

Description:
Compute the angle between two segments. This is used as part of the process to decide which arc to follow out of a node.

NAME	I/O	TYPE	LENGTH	DESCRIPTION
SEG1	I	I/R	4	Endpoints of segment 1 in form X1, Y1, X2, Y2
SEG2	I	I/R	4	Endpoints of segment 2 in form X1, Y1, X2, Y2
ANGLE	O	R	1	Returned angle in radians

Note: Assumption is that X2, Y2 of segment 1 = X1, Y1 of segment 2

Subroutine FORM

Description:

This routine scans the arc file, using arcs to create new features.

Subroutine ISLAND:

Description:

For area features, this routine assigns islands to parents.

Subroutine WEEDIT (XCOR, YCOR, NCOR1, NCOR2, TOL, IER)

Description:

This routine allows the user to weed, or thin, the coordinate data in an arc, at the minimum, it removes duplicate points.

NAME	I/O	TYPE	LENGTH	DESCRIPTION
XCOR	I	I/R	NCOR1	Input X coordinates for arc
YCOR	I	I/R	NCOR1	Input Y coordinates for arc
NCOR1	I	I/R	1	# of coordinate pairs in input arc
NCOR2	I	I/R	1	# of coordinate pairs in output arc
XCOR	O	I/R	NCOR2	X coordinates for weeded arc
YCOR	O	I/R	NCOR2	Y coordinates for weeded arc
IER	O	I	1	Error return 1 = normal

F.3.5 DBMS Shell

Subroutine DPOPN (DBNAME, FLAGS, NFLAGS, IER)

Description:

This routine opens an attributes database. It also loads the logical schema definition into the common block/ SCHEMA/

NAME	I/O	TYPE	LENGTH	DESCRIPTION
DBNAME	I	Character	Dynamic(1)	1 to 10 character database name
FLAGS	I	Integer	Dynamic(1)	Flags to be passed to DBMS as part of open. Some DBMS's, such as INGRES, allow flags. The integer codes for a flag are converted to internal codes inside this routine. For example, a code of 1 in an INGRES environment would be equal to T^U, which is the flag for enabling update of the system tables.
NFLAGS	I	Integer	1	Number of flags. Can be 0 RIM for example, has no flags.
IER	O	Integer	1	Error return 1 = normal Anything else is error on database open.

Subroutine CHKKEY (KEY, TYPE, LENGTH, LOCATE, IER)

Description:

This routine checks for the occurence of a key by using the logical schema for an attributes database.

Common Blocks

/SCHEME/ = Block storing schema definition

Useage

NAME	I/O	TYPE	LENGTH	DESCRIPTION
KEY	I	I	16	Field key to search for
TYPE	0	I	1	Data type - integer, real, character, date, time
LENGTH	0	I	1	Length of field in bytes
LOCATE	0	I	1	Field location in a row
IER	0	I	1	Error. 1 = normal, 0 = key not found

Subroutine DBGROW (RELNAM, RELNUM, KEY, ARRAY, IER)

Description:

This routine retrieves a row of data from the relational DBMS. The assumption is made that each map in the database has a unique relation name based on the map name. Further, each relation has one and only one table associated with it. This routine passes KEY into the required set of information to make the appropriate retrieval calls to the DBMS. The key is comprised of the map name and feature pointer. Within the routine, the actual work is performed by a set of calls to the DBMS. For RIM, the sequence is:

```
CALL RMWHER
CALL RMGET
```

For INGRES, the sequence is:

```
# RETRIEVE (Table name (n) = all)
# WHERE MAPID = KEY1 and FEATURE = KEY2
```

NAME	I/O	TYPE	LENGTH	DESCRIPTION
RELNAM	I	Character	10	Relation name
RELNUM	I	Integer	1	Relation number (should be 0 for now)
KEY	I	Character	dynamic	Search key
ARRAY	O	Integer	length of row in characters	Returned row of data
IER	O	Integer	1	Error return 1 = normal

Note: DBGROW and DBPROW have a common block that is loaded from

disk that contains the definition of the relational table for a given map. This definition must include the key (column) name, data type, and number of characters. This information is required to parse the search key for passage to the relational DBMS. The same definition could be used for any relational DBMS. The common block is called /SCHEMA/.

Subroutine DBPROW (RELNAM, RELNUM, KEY, ARRAY, IER)

Description:

This routine stores a row of data from the relational DBMS. The assumption is made that each map in the database has a unique relations name based on the map name. Further, each relation has one and only one table associated with it. This routine passes KEY into the required set of information to make the appropriate retrieval calls to the DBMS. The key is comprised of the map name and feature pointer. Within the routine, the actual work is performed by a set of calls to the DBMS. For RIM, the sequence is:

```
CALL RMWHER
CALL RMLOAD
```

For INGRES, the sequence is:

```
# LOADTABLE
# PUTROW
# UNLOADTABLE
```

NAME	I/O	TYPE	LENGTH	DESCRIPTION
RELNAM	I	Character	10	Relation name
RELNUM	I	Integer	1	Relation number (should be 0 for now)
KEY	I	Character	dynamic	Search key
ARRAY	O	Integer	length of row in characters	Row of data to store
IER	O	Integer	1	Error return 1 = normal

Note: DBGROW and DBPROW have a common block that is loaded from disk that contains the definition of the relational table for a given map. This definition must include the key (column) name, data type, and number of characters. This information is required to pause the search key for passage to the relational DBMS. The same definition could be used for any relational DBMS.

Subroutine DBGVAR (ARRAY, LEN, IVAL, RVAL, STRING, KEY, IER)

Description:

Given a row of data from a relation, this routine returns a specified field value. The actual retrieval is done by comparing the field ID with what is stored in the common block containing the schema for a row in the relation. This common block is described in the Common block section and in Subroutine RELCOM.

NAME	I/O	TYPE	LENGTH	DESCRIPTION
ARRAY	I	Int	LEN	Array containing row of information
LEN	I	Int	1	Length of array
IVAL	O	Int	1	Word to return integer value in
RVAL	O	Real	1	Location to return real value in
STRING	O	Character	dynamic	Array to return character string in
KEY	I	Int	1	Field to return
IER	O	Int	1	Error return 1 = normal

Subroutine DBPVAR (ARRAY, LEN, IVAL, RVAL, STRING,
KEY, ITYPE, IER)

Description:

Given a row of data from a relation, this routine replaces a specified field value. The actual replacement is done by comparing the field ID with what is stored in the common block containing the schema for a row in the relation. This common block is described in the Common block section and in Subroutine RELCOM.

NAME	I/O	TYPE	LENGTH	DESCRIPTION
ARRAY	I	Int	LEN	Array containing row of information
LEN	I	Int	1	Length of array
IVAL	I	Int	1	Word to send integer value in
RVAL	I	Real	1	Location to send real value in
STRING	I	Character	dynamic	Array to send character string in
KEY	I	Int	dynamic	Field to replace
IER	O	Int	1	Error return 1 = normal
ITYPE	I	Int	1	Data type 1= integer 2= real 3= character

Subroutine DBRDEL (NUMBER, NAME, KEY, IER)

Description

This routine deletes a row of attribute data from a relation. The relation number and name set the DBMS pointer at therelation. The attribute key is used to position the pointer at the correct row. The sequence of RIM calls would be.

```
CALL RMFIND (NUMBER, NAME)
CALL RMWHER (NUMBER, 3HKEY, 3HEQS, KEY 1,0,1)
CALL RMGET (NUMBER, ARRAY)
CALL RMDEL (NUMBER)
```

Useage

NAME	I/O	TYPE	LENGTH	DESCRIPTION
NUMBER	I	I	1	Relation number
NAME	I	Character	Dynamic	Relation name
KEY	I	Character	Dynamic	Attribute key
IER	O	I	1	Error return

Subroutine DBSEAR (STRING, NFEATS, IER)

Description

This routine takes a MOSS boolean attribute query, converts to reverse polish notation (using subroutine RPN), and converts it to the appropriate format for a call to the DBMS query function. The routine returns a list of Featrue keys for those features whose attributes satisfied the query. The feature keys are returned via a disk file.

For example, say we are going to use RIM. Assume a MOSS query

```
ATT1 EQ 4 OR ATT1 EQ7 OR ATT1 EQ 12 OR &
ATT2 EQ "STRING" AND ATT3 GT 5. AND ATT3 LT 10 *
```

Would be translated into a series of arrays in the form:

```
attname (1)    = 8HATT1
attname (2)    = 8HATT2
attname (3)    = 8HATT3
attname (4)    = 8HATT3
operator (1)   = 4HEQ
operator (2)   = 4HEQS
operator (3)   = 4HGT
operator (4)   = 4HEQA
value (1,1)    = 4
value (1,2)    = 7
value (1,3)    = 12
value (1,4)    = value (1,5) = 0
value (2,1)    = 11
value (2,2)    = 0
value (2,3)    = 4HTEXT
value (2,4)    = 4H STR
value (2,5)    = 4HING
```

```
value (3,1)    = 5.
value (3,2)    = value (3,3) = value (3,4) = value (3,5) = 0
value (4,1)    = 4HATT4
value (4,2)    = value (4,3) = value (4,4) = value (4,5) = 0
numval (1)     = 3
numval (2)     = 1
numval (3)     = 1
numval (4)     = 1
nextboo (1)    = 4HOR
nextboo (2)    = 4HAND
nextboo (3)    = 4HAND
nextboo (4)    = 0
numboo         = 4
```

For a call to RMWHER.

Useage

NAME	I/O	TYPE	LENGTH	DESCRIPTION
STRING	I	Character	Dynamic	MOSS Attribute Query
NFEATS	O	Integer	1	# of features meeting search criteria
IER	O	Integer	1	Error return

Subroutine DBSLST (IOUT, IFLAG, IER)

Description:

This routine lists pertinent fields in a logical schema for an attributes database. IFLAG controls what parts of the schema are to be listed. The schema is stored in a common block called /SCHEMA/ and in a disk file called THEME.RL, where THEME is the THEME or map name that a logical schema is associated with. This file is created by the program ATTDESCRIBE if you are using MOSS or the SCHEMA builder if using a relational DBMS.

Useage

NAME	I/O	TYPE	LENGTH	DESCRIPTION
IOUT	I	INT	1	Logical unit for output
IFLAG	I	INT	1	Flag indicating one of the following: 1 = keys only 2 = keysand descriptions 3 =keys, descriptions, and type 4 = keys, description, type and length 5 = keys, description, type,length, and field relation
IER	O	INT	1	Error return

Subroutine DUMPAT (ICHAN1, ICHAN2, IER)

Description:

This routine dumps MOSS attributes file to an ASCII text file on disk. The logical schema file is used to parse the internal format back into the original text input format.

Useage:

NAME	I/O	TYPE	LENGHT	DESCRIPTION
ICHAN1	I	INTEGER	1	Logical unit number for attributes file
ICHAN2	I	INTEGER	1	Logical unit number for output ASCII file
IER	O	INTEGER	1	Error return. 1 = normal

Common Blocks

/SCHEMA/ Logical schema definition for attributes file

Subroutine RELCOM (IER)

Description:

This routine reads the schema for a row of data in a relation. This schema contains information on the field name, field ID, field type, and field length. It may be thought of as the cross reference index between whats in the DBMS and what MOSS actually recognizes.

NAME	I/O	TYPE	LENGTH	DESCRIPTION
IER	O	Int	1	Error return 1= normal

Common Block

/SCHEMA /NAME(200,6), ID(200), TYPE(200), LENGTH(200), NVARS

NAME	TYPE	LENGTH	DESCRIPTION
NAME	Int	1200 words	This array contains up to 200 key words for fields in a row in the relation
ID	Int	200	Field ID's that correspond to the field names
TYPE	Int	200	Data type 1= integer, 2= real, 3= character
LENGTH	Int	200	Field length in bytes
NVARS	Int	1	# of fields in Row

Special Files

This routine reads from a file called THEME.RL, where THEME is the name of an actual theme in MOSS, such as SOILS. This file

contains the schema for the attributes related to a theme. It is a sequential file.

F.3.7 Database primitives

This section describes the arc/node database primitives.

Subroutine OPNMAP (CHANEL, MAPNAM, WHTFIL, FLAG, OPTIONN, IER)

Description:

This routine will open one of the arc/node files for a map. The proper extension is added to the map name and the appropriate file opened and/or created.

NAME	I/O	TYPE	LENGTH	DESCRIPTION
CHANEL	I	Int	1	Inputlogical unit number
MAPNAM	I	Int or character	16	Map name
WHTFIL	I	Int	1	Which file to actually open for map. 1= map header 2= feature file 3= arc file 4= node file 5= coordinate file 6= feature pointer file 7= attribute file 8= node index file 9= cartographic file 10= fast file
FLAG	I	Int	1	Access node 1= read 2= write 3= read/write
OPTION	I	Int	1	How to open 1= open only

				2= create and open 3= delete, create, open
IER	I	Int	1	Error return 1= normal

Note: The file lengths are retrieved from the PARAM common block.

CALLED ROUTINES GTNAM, ERMES, system open

Subroutine DELMAP (MAPNAM, IER)

Description:

This routine deletes a map from the database. The entry is removed from the map names file and all corresponding disk files are deleted. Please note that this is an upgrade to the existing MOSS DELMAP subroutine.

NAME	I/O	TYPE	LENGTH	DESCRIPTION
MAPNAM	I	Int	16	Map name to delete
IER	O	Int	1	Error return 1= normal

Called Routines: DFILW

Subroutine CREMAP (MAPNAM, IER)

Description:
This routine creates all the files necessary for a new arc/node map in MOSS. The number of bytes per physical record are stored in the common block /PARAM/

NAME	I/O	TYPE	LENGTH	DESCRIPTION
MAPNAM	I	Int	16	Map name to create
IER	O	Int	1	Error return 1= normal

Called Routines GTNAM, ERMES, CFILW

Subroutines RENAME (OLDNAM, NEWNAM, IER)

Description:

This routine allows the user to rename a MOSS arc/node file. All the disk files are renamed and the map names entry is changed.

NAME	I/O	TYPE	LENGTH	DESCRIPTION
OLDNAM	I	character/Int	16	Old map name
NEWNAM	I	character/Int	16	New map name
IER	O	Int	1	Error return 1= normal

Called Routines GTNAM, TUPDIR, DELFIL, ERMES

Subroutine CRENOD (CHANEL, XCOR, YCOR, NODEID, IER)

Description:

This routine creates a node and adds the entry to the node file.

NAME	I/O	TYPE	LENGTH	DESCRIPTION
CHANEL	I	Int	1	Logical unit # for node file
XCOR	I	Int	1	X coordinate for node
YCOR	I	Int	1	Y coordinate for node
NODID	O	Int	1	Returned node ID
IER	O	Int	1	Error return 1= normal

Subroutine UPNODE (CHANEL,NODEID,ARCID, WHICH, IER)

Description:

This routine adds an arc reference or deletes an arc reference from a node. If there are more than nine arc references, then this routine performs the shuffle required to allow for nine for arc entries. This procedure is described in section 2 under the discussion on the Node file structure.

NAME	I/O	TYPE	LENGTH	DESCRIPTION
CHANEL	I	Int	1	Logical unit # fornode file
NODEID	I	Int	1	Node ID of node to be updated
ARCID	I	Int	1	Arc pointer to be added to list of arcs
WHICH	I	Int	1	0= add arc, 1= delete arc
IER	O	Int	1	Error return 1=normal

Subroutine DELNOD (CHANL1, CHANL2, CHANL3, NODEID, IER)

Description:

This routine deletes all the edges associatedwith the node and then deletes the node record from the nodefile. Deletion is done by setting the node ID to the largest negative integer number allowed in the system.

NAME	I/O	TYPE	LENGTH	DESCRIPTION
CHANL1	I	Int	1	Chanel number of node file
CHANL2	I	Int	1	Channel number of arc file
CHANL3	I	Int	1	Channel number for coordinate file
NODEID	I	Int	1	ID of node to be deleted
IER	O	Int	1	Error return 1= normal

Subroutine LSTNOD (CHANEL, NODEID, IOUT, IER)

Description:

This routine lists the contents of a node record

NAME	I/O	TYPE	LENGTH	DESCRIPTION
CHANEL	I	Int	1	Logical unit number for node file
NODEID	I	Int	1	ID of node record to list
IOUT	I	Int	1	Logical unit number for list file
IER	O	Int	1	Error return 1= normal

Subroutine FNDNOD (CHANEL, XCOR, YCOR, TOL, NODEID, IER)

Description:

Given a search coordinate, this routine finds the node closest to it and returns the node ID.

NAME	I/O	TYPE	LENGTH	DESCRIPTION
CHANEL	I	Int	1	Logical unit number for node file
XCOR	I	Int	1	X coordinate for search point
YCOR	I	Int	1	Y coordinate for search point
TOL	I	Real	1	Search tolerance (can be 0 for exact match)
NODEID	O	Int	1	Returned node ID
IER	O	Int	1	Error return 1= normal

Subroutine ADDARC (CHANEL, COSTAR, COSTOP, NODESR, NODESP, IER)

Description:

This routine adds an entry to the arc file. The routines ADDCOR and UPNODE should be called first.

NAME	I/O	TYPE	LENGTH	DESCRIPTION
CHANEL	I	Int	1	Logical unit number for arc file
COSTAR	I	Int	1	Coordinate start record
COSTOP	I	Int	1	Coordinate stop record
NODESR	I	Int	1	Start node ID
NODESP	I	Int	1	Stop node ID
IER	O	Int	1	Error return 1= normal

Subroutine DELARC (CHANEL, ARCPT, NODESR, NODEST, IER)

Description:

This routine deletes an arc from the arc file. This routine should be called before UPNODE to update the node.

NAME	I/O	TYPE	LENGTH	DESCRIPTION
CHANEL	I	Int	1	Logical unit number for arc file
ARCPT	I	Int	1	Arc pointer
NODESR	O	Int	1	Start node ID
NODEST	O	Int	1	Stop node ID
IER	O	Int	1	Error return 1= normal

Subroutine FNDARC (CHAN1, CHAN2, CHAN3, XCOR, YCOR, ZCOR, ARCID, IER)

Description:

This routine searches the arc file for the arc closest to the search point (XCOR, YCOR, ZCOR). The method is to scan the fast file to determine potential candidates then to do a point to arc distance calculation.

NAME	I/O	TYPE	LENGTH	DESCRIPTION
CHAN1	I	Int	1	Logical unit number for arc file
CHAN2	I	Int	1	Logical unit number for fast file
CHAN3	I	Int	1	Logical unit number for coordinate file
XCOR	I	Real	1	X coordinate of search point
YCOR	I	Real	1	Y coordinate of search point
ZCOR	I	Real	1	Z coordinate of search point
ARCID	I	Int	1	Returned arc pointer
IER	I	Int	1	Error return 1= normal

Subroutine LSTARC (ICHAN, EDGID, IOUT, IER)

Description:

Dump contents of edge record to some output device.

Useage:

NAME	I/O	TYPE	LENGTH	DESCRIPTION
ICHAN	I	INT	1	Logical unit number for edge file
EDGID	I	INT	1	Edge ID to list
IOUT	I	INT	1	Logical unit number of list device
IER	O	INT	1	Error return

Subroutine UPARC (ICHAN, EDGID, FID, IDO, IEX, IER)

Description:

This routine updates the feature keys in an edge record. This would happen during feature formation or feature deletion.

Useage:

NAME	I/O	TYPE	LENGTH	DESCRIPTION
ICHAN	I	INT	1	Logical unit number for edge file
EDGID	I	INT	1	Edge ID to update
FID	I	INT	1	Feature key to add to edge record
IDO	I	INT	1	1 = left, 2 = center, 3 = right
IEX	I	INT	1	Reserved
IER	O	INT	1	Error return

Subroutine ADDCOR (ICHAN, COREID, EDGE/NODEID, XYZ, NPTS, TYPE, IER)

Description:

This routine adds a record to the coordinate file.

Useage:

NAME	I/O	TYPE	LENGTH	DESCRIPTION
ICHAN	I	INT	1	Logical unit number for coordinate file
COREID	I	INT	1	Coordinate ID or physical record number
EDGID/NODEID	I	INT	1	Edge or Node ID
X,Y,Z	I	INT	NPTS	X, Y, Z coordinate triples
NPTS	I	INT	1	#Coordinate triples

Subroutine DELFET (ICHAN1, ICHAN2, ICHAN3, FID, IER)

Description:
This subroutine deletes a feature and all of its associated entries in the edge and chain files.

Routines Called:

DELARC, CNGATT

Useage:

NAME	I/O	TYPE	LENGTH	DESCRIPTION
ICHAN1	I	INT	1	Logical unit feature file
ICHAN2	I	INT	1	Logical unit chain file
ICHAN3	I	INT	1	Logical unit edge file
FID	I	INT	1	Feature ID of feature to delete
IER	I	INT	1	Error return

Subroutine BLDFET (ICHAN1, ICHAN2, ICHAN3, ICHAN4, FID, NPTS, NSLAND, IER)

Description:

This routine uses a feature ID to access the feature chain file to build arrays of (x,y) coordinate pairs to be returned to calling routines. This is the routine most required by current MOSS to access the arc/node data structure.

Useage:

NAME	I/O	TYPE	LENGTH	DESCRIPTION
ICHAN1	I	INT	1	Logical unit chain file
ICHAN2	I	INT	1	Logical unit feature file
ICHAN3	I	INT	1	Logical unit edge file
ICHAN4	I	INT	1	Logical unit coordinate file
FID	I	INT	1	Feature file
NPTS	O	INT	1	# coordinate pairs in feature
NSLAND	O	INT	1	# islands in feature
IER	O	INT	1	Error return

APPENDIX G

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

                     NZ IS AN OUTPUT ARGUMENT, IT INDICATES WHAT FURTHER
                     IS REQUIRED


                     NZ=0......THE MAJOR AND MINOR POLYGONS HAVE BOUNDARI
                               THAT INTERSECT. THE NEW POLYGON MUST BE BU
                               USING  THE INTERSECTION LIST.
                     NZ=1......THE EXTERIOR BOUNDARY OF THE MAJOR POLYGON
                               COMPLETELY WITHIN THE MINOR POLYGON. THE M
                               POLYGON HAS BEEN USED AS A BASIS FOR BUILD
                               THE NEW POLYGON AND HAS BEEN WRITTEN OUT.
                               FURTHER PROCESSING OF THIS MAJOR POLYGON I
                     NZ=2......NO FURTHER PROCESSING OF THIS MINOR POLYGO
                               NECESSARY BECAUSE EITHER   1)THE TWO POLYG
                               DO NOT INTERSECT OR 2) THE EXTERIOR BOUNDA
                               THE MINOR POLYGON IS COMPLETELY INSIDE THE
                               POLYGON  THEREFORE, THE MINOR POLYGON HAS
                               AS A BASIS FOR BUILDING THE NEW POLYGON AN
                               BEEN WRITTEN OUT
                     NZ=-1.....ERROR - AN ODD NUMBER OF INTERSECTIONS HAS


  REVISED BY CARL REED, AUTOMETRIC INC, DEC. 28, 1981

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

      COMMON /O/ XMIN,XMAX,YMIN,YMAX,PMIN,PMAX,QMIN,QMAX
      DOUBLE PRECISION X1,Y1,X2,Y2,SLXY,P1,P2,Q1,Q2,TX,BX,TY,BY,
     1   DX,DY,BP,TP,BQ,TQ,DP,DQ,SLPQ,TM1,TM2,TM3,TM4,A,
     2   RSLXY,RSLPQ,XINT,YINT
      LOGICAL SAME
      INCLUDE 'ARC.COMMON'
      DIMENSION IS(NDIM),JS(NDIM)
      NZ=0

       IF(X(1).NE.P(1)) GO TO 2
       IF(Y(1).NE.Q(1)) GO TO 2
       IF(N.NE.M)       GO TO 2
       DO 1 I=2,N
       IF(X(I).NE.P(I)) GO TO 2
       IF(Y(I).NE.Q(I)) GO TO 2
1      CONTINUE
       NZ=4
       RETURN
2      CONTINUE

  THE NUMBER OF INTERSECTIONS IN A LIST

      NS=0



                     FIND ALL INTERSECTIONS BETWEEN BOUNDARIES OF THE MAJ
                     MINOR POLYGONS AND MAKE A LIST OF    1) THE INTERSEC
```

Figures for Design
Document

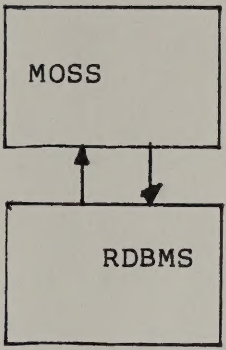
Table of Contents
Figures

1.1 RDBMS Interface Boundaries
1.2
2.2 NODE Record Layout
2.3 EDGE Relation to Other Files
2.4 Coordinate File Layout
2.5 Feature File Layout
2.6 Feature Chain File Layout
2.7 Attribute File Layout for Non-RDBMS Implementation
2.9 Layout for Index File
2.10 Work and Master Headers
3.1 Attribute Reclassification
3.2 Attribute Entry and Validation
3.4 Attribute Delete
3.5 Attribute Merge
3.6 Attribute Dump to Interchange File
3.7 Edge Match Process
3.8 Determine Adjacent Sheets
3.9 Edge Match Processor
3.10 User Interface to Overlay Process
3.11 Spatial Pre-processor to Determine Candidates
3.12 Intersection Processor
3.13 Overlaying Map A on Map B
3.14 Build New Arc/Node Map from Intersection List
3.15 Attribute Build Processor
3.16 Sliver Removal
3.18 Zone Generation User Interface
3.19 Zone Generation
3.20 Polygon to Arc User Interface
3.21 Pre-Processor to Find Candidate for Polygon to Arc
3.22 Edge Intersection
3.23 Open Attributes Database
3.25 Retrieve a Row from a Table
3.27 Put Value into Field Location in Row of Attribute Data
3.29 Attribute Retrieval

Figure 1.1

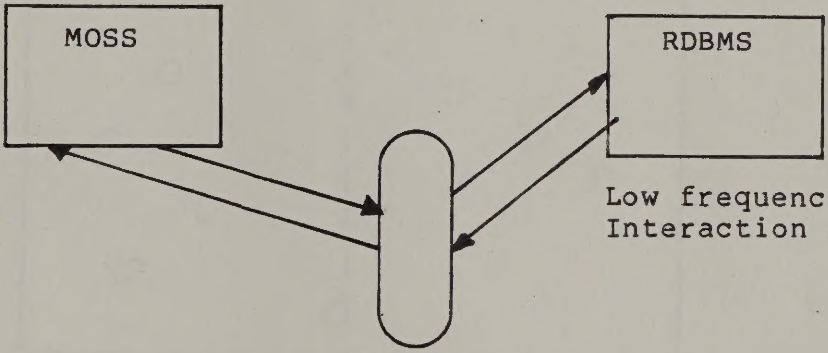
RDBMS Interface Boundaries

Total Integration



MOSSis totally dependent on DBMS. Pull it out and MOSS collapses.

Complete Separation



Low frequency level Interaction only

Auxilliary file

DBMS Shell/Direct Access

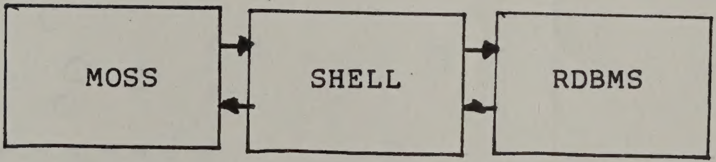
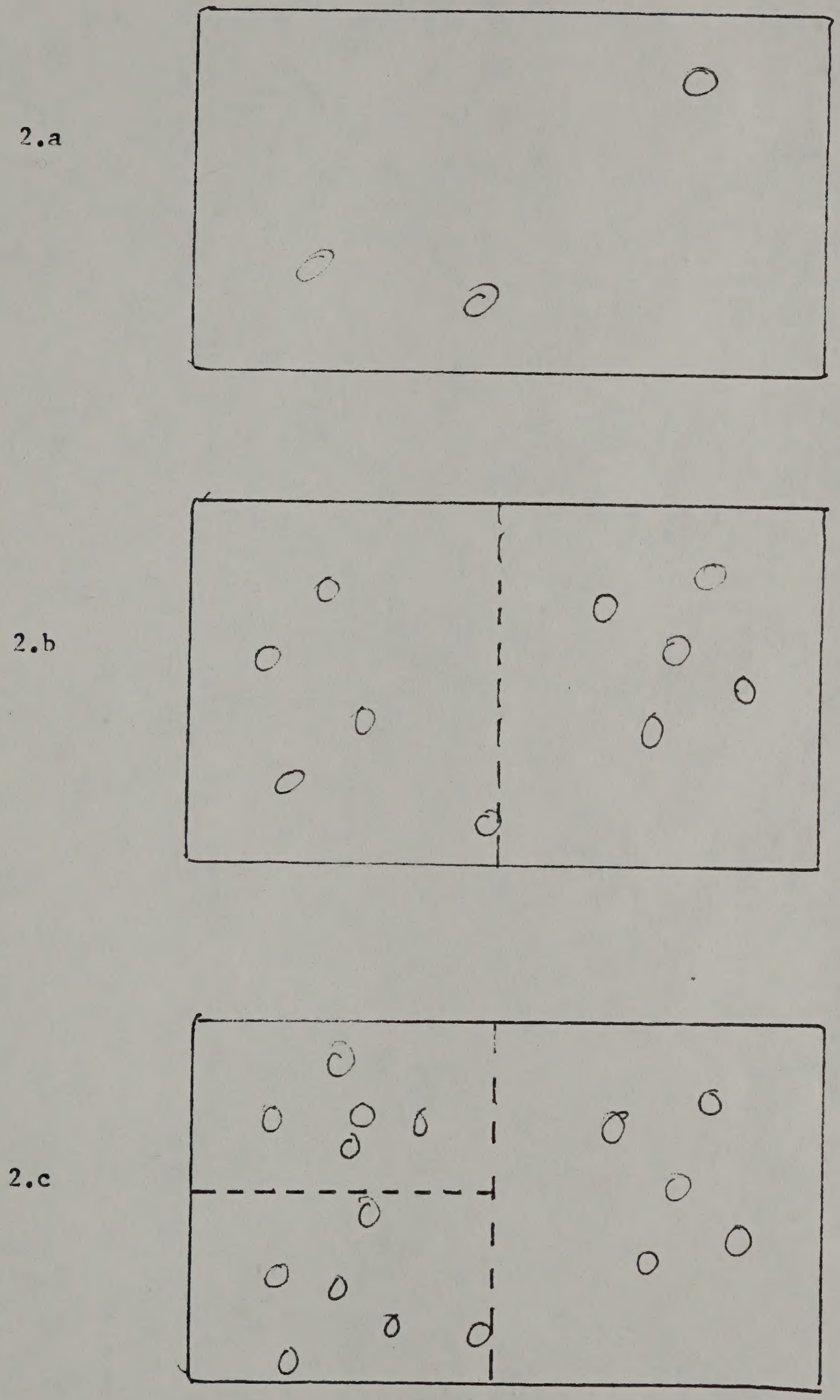
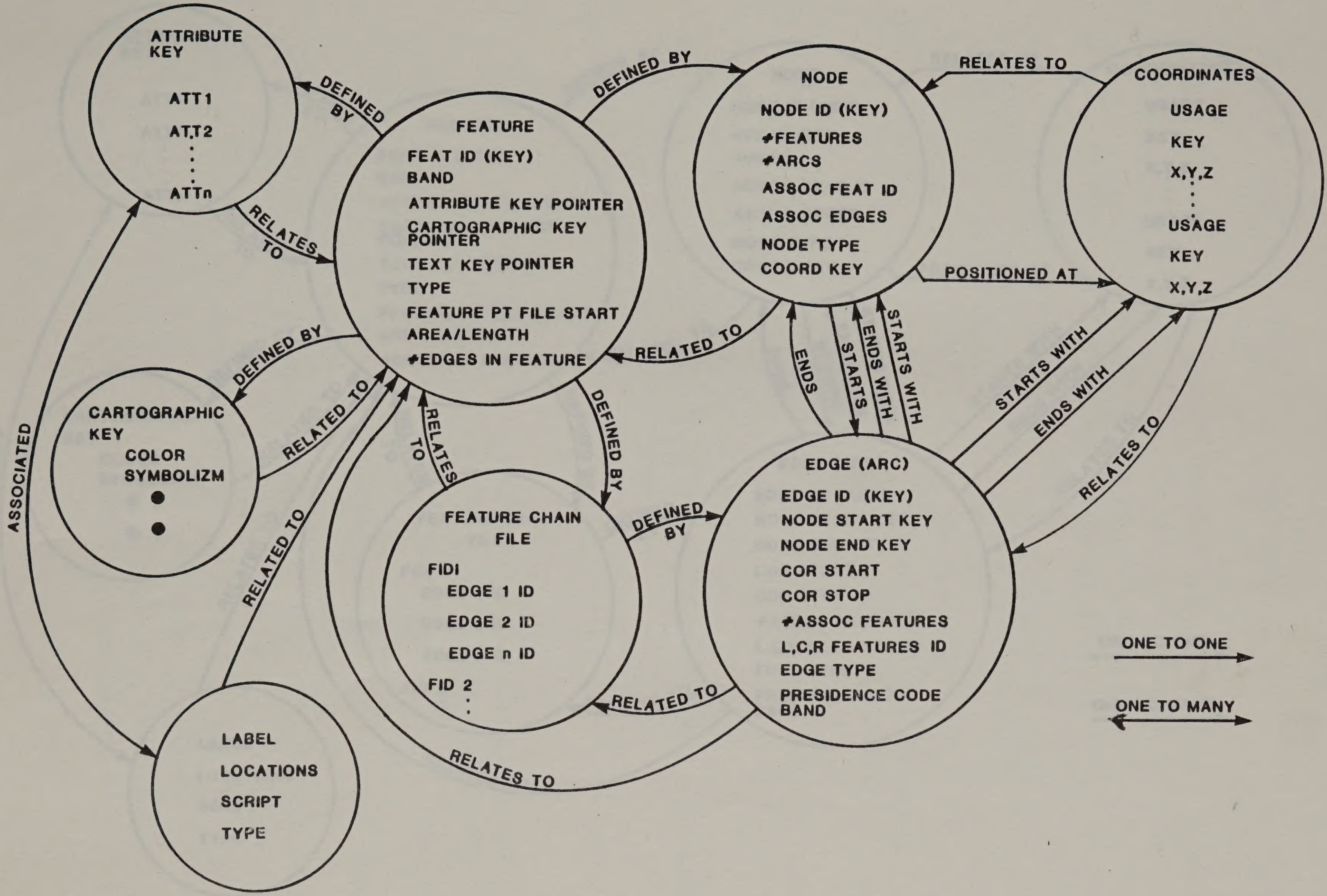

MOSS
SHELL
RDBMS

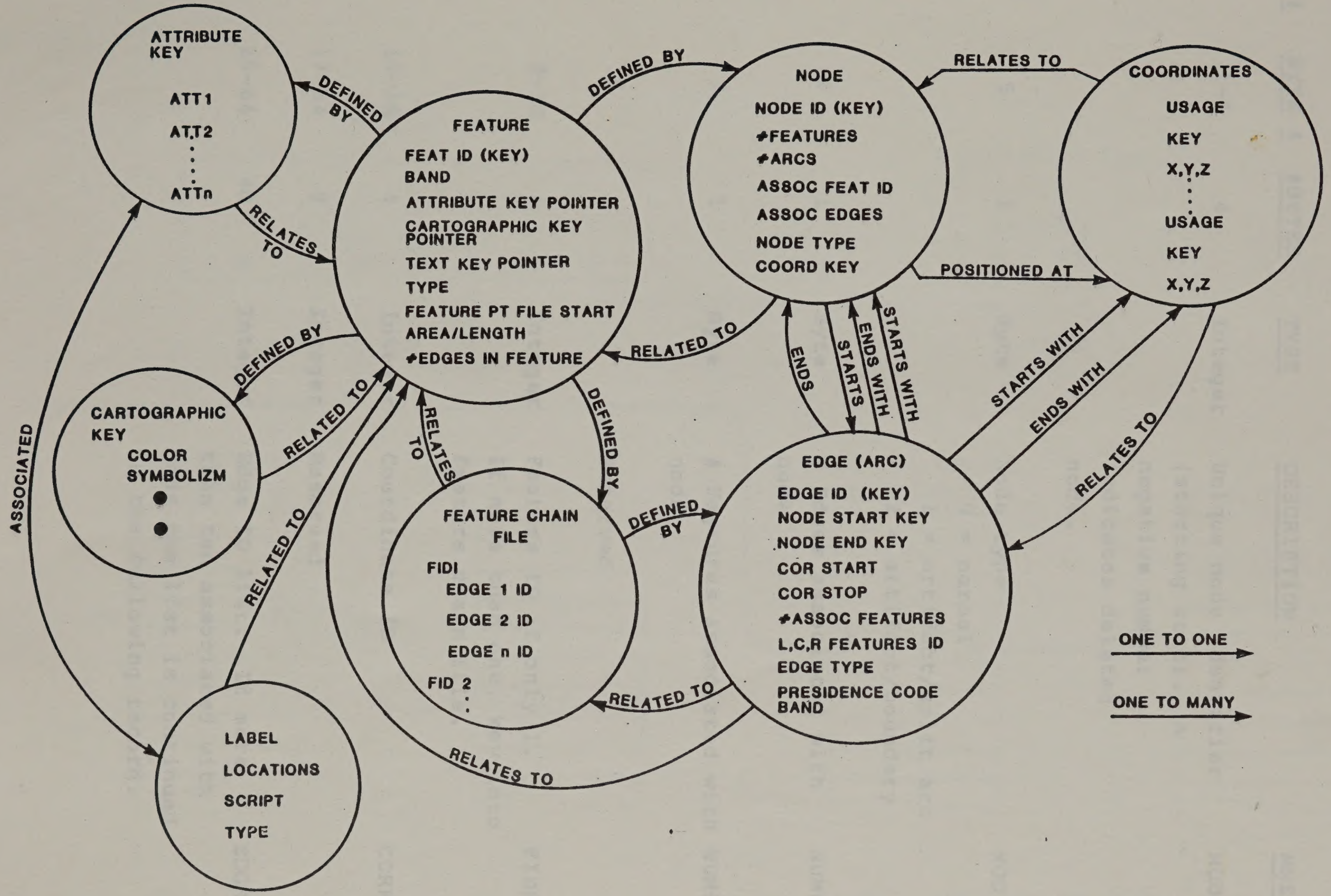
Figure 1.2



Dynamic Tiling


ATTRIBUTE KEY
ATT1
AT,T2
ATTn
FEATURE
FEAT ID (KEY)
BAND
ATTRIBUTE KEY POINTER
CARTOGRAPHIC KEY POINTER
TEXT KEY POINTER
TYPE
FEATURE PT FILE START
AREA/LENGTH
#EDGES IN FEATURE
NODE
NODE ID (KEY)
#FEATURES
#ARCS
ASSOC FEAT ID
ASSOC EDGES
NODE TYPE
COORD KEY
COORDINATES
USAGE
KEY
X,Y,Z
USAGE
KEY
X,Y,Z
CARTOGRAPHIC KEY
COLOR
SYMBOLIZM
●
●
FEATURE CHAIN FILE
FIDI
EDGE 1 ID
EDGE 2 ID
EDGE n ID
FID 2
EDGE (ARC)
EDGE ID (KEY)
NODE START KEY
NODE END KEY
COR START
COR STOP
#ASSOC FEATURES
L,C,R FEATURES ID
EDGE TYPE
PRESIDENCE CODE
BAND
LABEL
LOCATIONS
SCRIPT
TYPE
DEFINED BY
RELATES TO
DEFINED BY
RELATES TO
DEFINED BY
RELATED TO
POSITIONED AT
ENDS
STARTS
ENDS WITH
STARTS WITH
STARTS WITH
ENDS WITH
RELATES TO
DEFINED BY
RELATED TO
RELATES TO
DEFINED BY
DEFINED BY
RELATED TO
RELATES TO
ASSOCIATED
RELATED TO
ONE TO ONE
ONE TO MANY

UDS06


ATTRIBUTE KEY
ATT1
ATT2
ATTn
DEFINED BY
RELATES TO
FEATURE
FEAT ID (KEY)
BAND
ATTRIBUTE KEY POINTER
CARTOGRAPHIC KEY POINTER
TEXT KEY POINTER
TYPE
FEATURE PT FILE START
AREA/LENGTH
#EDGES IN FEATURE
DEFINED BY
NODE
NODE ID (KEY)
#FEATURES
#ARCS
ASSOC FEAT ID
ASSOC EDGES
NODE TYPE
COORD KEY
RELATES TO
COORDINATES
USAGE
KEY
X,Y,Z
USAGE
KEY
X,Y,Z
POSITIONED AT
RELATED TO
ENDS
STARTS
ENDS WITH
STARTS WITH
STARTS WITH
ENDS WITH
RELATES TO
ASSOCIATED
DEFINED BY
CARTOGRAPHIC KEY
COLOR
SYMBOLIZM
●
●
RELATED TO
RELATED TO
RELATES TO
DEFINED BY
FEATURE CHAIN FILE
FIDI
EDGE 1 ID
EDGE 2 ID
EDGE n ID
FID 2
DEFINED BY
EDGE (ARC)
EDGE ID (KEY)
NODE START KEY
NODE END KEY
COR START
COR STOP
#ASSOC FEATURES
L,C,R FEATURES ID
EDGE TYPE
PRESIDENCE CODE
BAND
RELATED TO
RELATES TO
LABEL
LOCATIONS
SCRIPT
TYPE
ONE TO ONE
ONE TO MANY

UDS06

Figure 2.2

Node Record Layout

WORD #	BYTE #	#BYTES	TYPE	DESCRIPTION	MNEMONIC
1	1-4	4	Integer	Unique node identifier (starting at 1). A negative number indicates deleted node.	NODEID
2	5	1	Byte	Node type 0 = normal 1 = artifact/split arc 2 = artifact/boundary	NODYTP
2	6	1	Byte	# Edges associated with node	NUMEDG
2	7	1	Byte	# Features associated with node	NUMFEA
2	8	1		Reserved	
3	9-12	4	Integer	Feature ID if only 1. If more than one, key into feature chain file.	FIDKEY
4	13-16	4	Integer	Coordinate ID	CORKEY
5-6	17-24	8	Integer	Reserved	
7-16	25-64	40	Integer	Edge ID list. If more than ten associated with node the list is continued on the following record.	EDGKEY

RELATION TO OTHER FILES

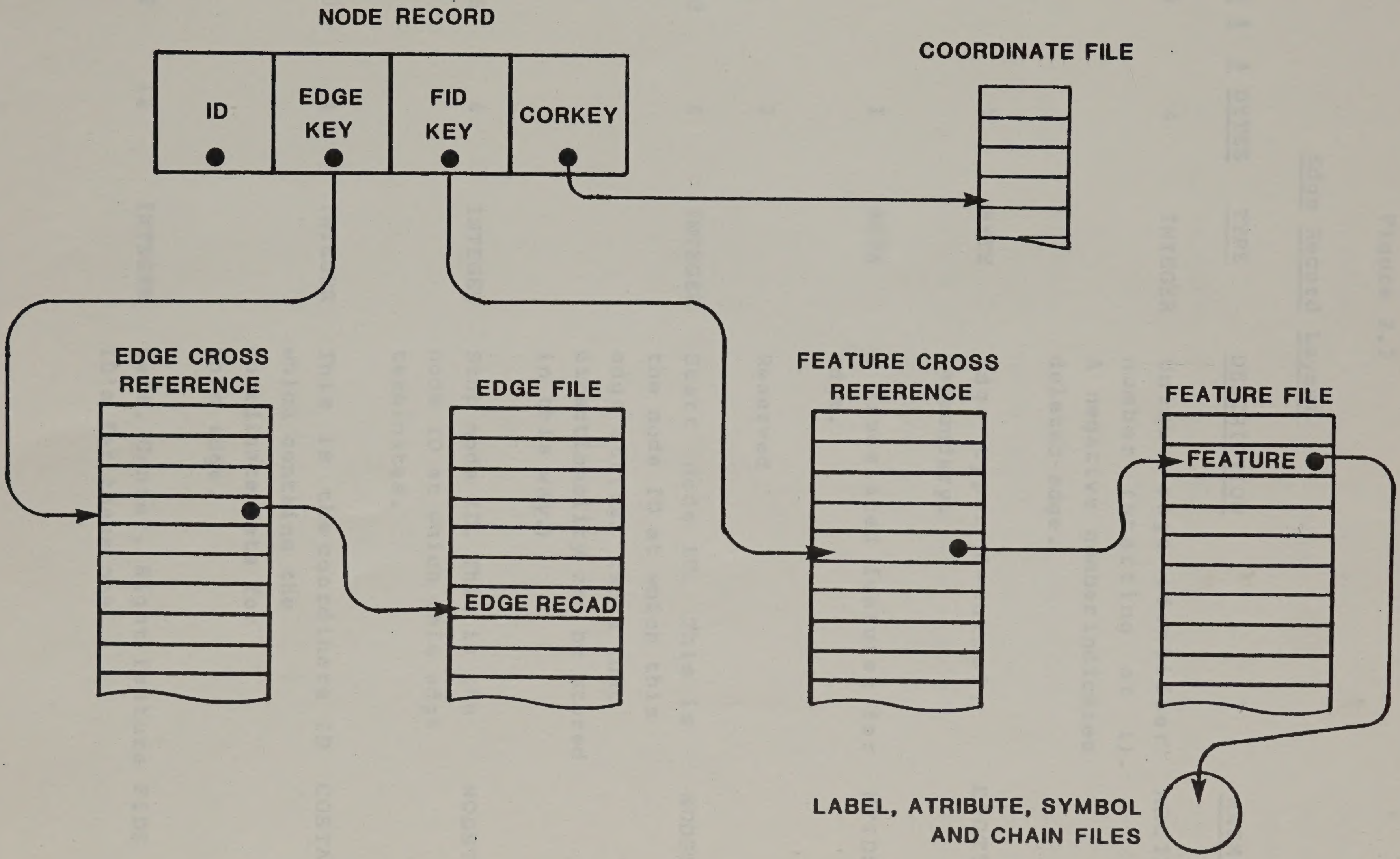

NODE RECORD
ID
EDGE KEY
FID KEY
CORKEY
COORDINATE FILE
EDGE CROSS REFERENCE
EDGE FILE
EDGE RECAD
FEATURE CROSS REFERENCE
FEATURE FILE
FEATURE
LABEL, ATRIBUTE, SYMBOL AND CHAIN FILES

Figure 2.3

Edge Record Layout

WORD #	BYTE #	# BYTES	TYPE	DESCRIPTION	MNEMONIC
1	1-4	4	INTEGER	Unique edge identifier number (starting at 1). A negative number indicates deleted edge.	ARCID
2	5	1	BYTE	Edge type. 0=normal, 1=boundary.	EDGTYP
2	6	1	BYTE	# Associated features for edge.	NFIDS
2	7-8	2		Reserved	
3	9-12	4	INTEGER	Start node ID. This is the node ID at which this edge starts. (Note that directionality can be stored in this way.)	NODSTAR
4	13-16	4	INTEGER	Stop node ID. This is the node ID at which this edge terminates.	NODSTOP
5	17-20	4	INTEGER	This is the coordinate ID which contains the coordinate data for this edge.	COSTAR
6-8	21-32	12	INTEGER	Left, Center, Right feature ID's for theme one.	FIDS

9-11	33-44	12	INTEGER	Minimum X, Y, and Z value for edge.	XMIN YMIN ZMIN
12-14	45-56	12	INTEGER	Maximum X, Y, and Z value for edge.	XMAX YMAX ZMAX
15	57-60	4	INTEGER	X Band (see description in text).	XBAND
16	61-64	4	INTEGER	Y Band (see description in text).	YBAND

COORDINATE FILE RELATIONSHIP TO OTHER FILES

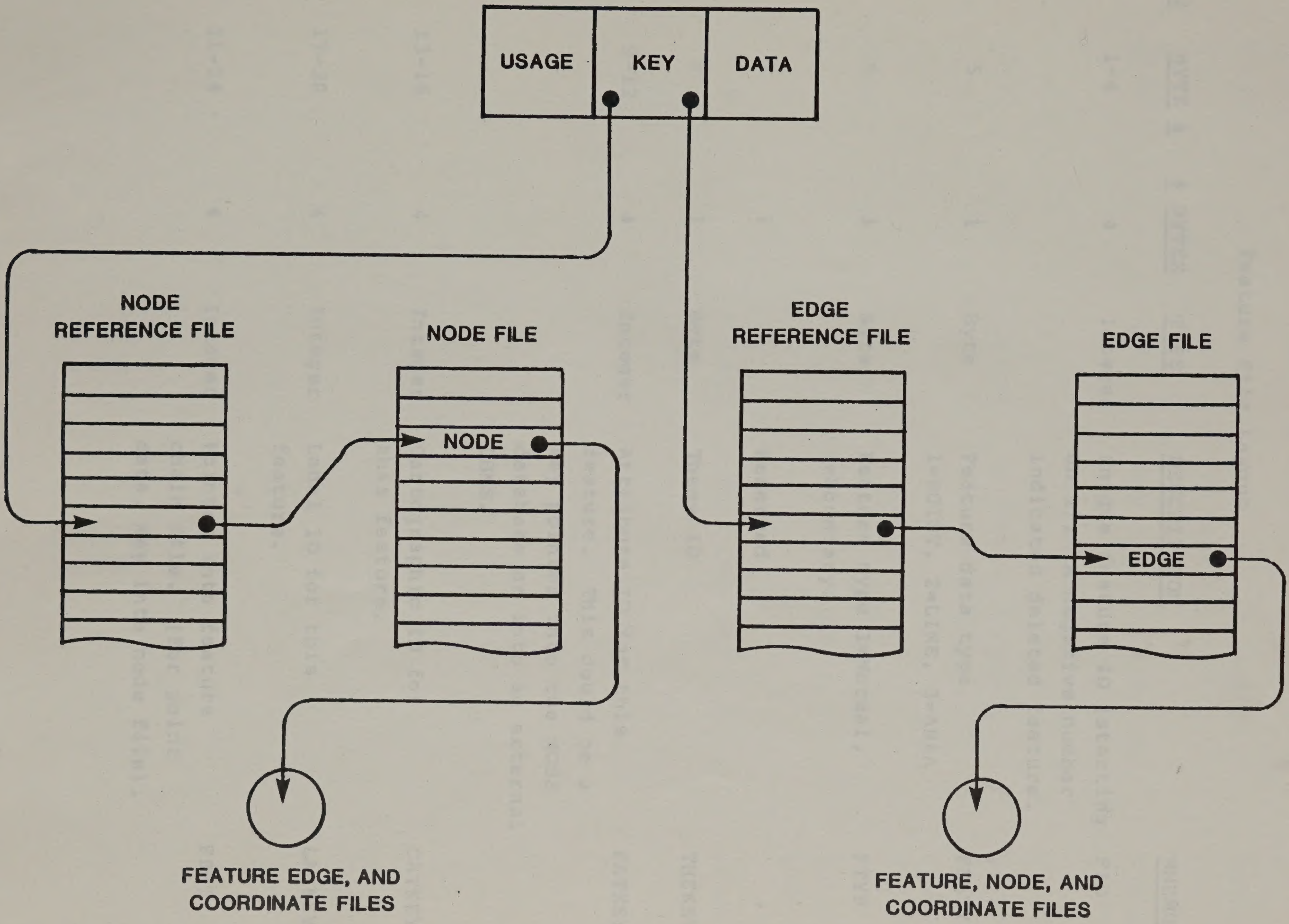

USAGE
KEY
DATA
NODE REFERENCE FILE
NODE FILE
NODE
EDGE REFERENCE FILE
EDGE FILE
EDGE
FEATURE EDGE, AND COORDINATE FILES
FEATURE, NODE, AND COORDINATE FILES

Figure 2.5

Feature File Layout

WORD #	BYTE #	# BYTES	TYPE	DESCRIPTION	MNEMONIC
1	1-4	4	Integer	Unique feature ID (starting on 1). A negative number indicates deleted feature.	FID
2	5	1	Byte	Feature data type 1=POINT, 2=LINE, 3=AREA	FEATYP
2	6	1	Byte	Feature type 1=Normal, 2=Boundary.	FTYP
2	7	1		Reserved	
2	8	1	Byte	Theme ID	THEKEY
3	9-12	4	Integer	Attribute ID for this feature. This could be a key pointer into the MOSS database or into an external DBMS.	FATKEY
4	13-16	4	Integer	Cartographic ID for this feature.	CATKEY
5	17-20	4	Integer	Label ID for this feature.	LABKEY
6	21-24	4	Integer	Pointer into feature chain file. (For point data, key into node file).	FPST

7	25-26	2	Integer	# Edges comprising feature	NEDGE
7	27-28	2	Integer	# Islands in feature for data type area.	NSLAND
8	29-32	4	Real	Area for area data and length for lineal data. Reserved for point.	ARELEN
9-11	33-44	12	Integer	Minimum X, Y and Z value for feature.	XMIN YMIN ZMIN
12-14	45-56	12	Integer	Maximum X, Y, and Z value for feature.	XMAX YMAX ZMAX
15	57-60	4	Integer	X Band (see description in text).	XBAND
16	61-64	4	Integer	Y Band (see description in text).	YBAND

EDGE FILE RELATION TO OTHER FILES

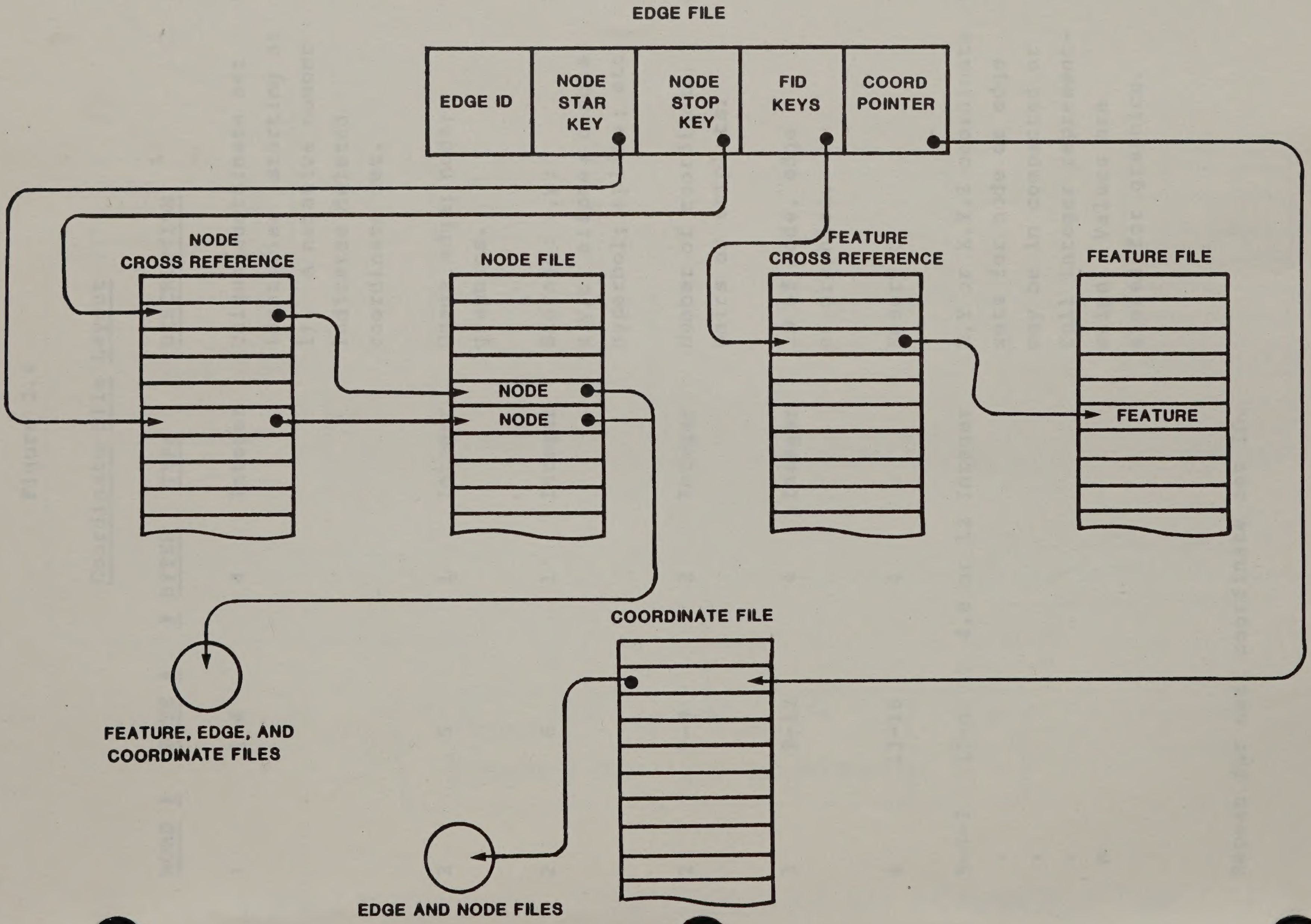

EDGE FILE
EDGE ID
NODE STAR KEY
NODE STOP KEY
FID KEYS
COORD POINTER
NODE CROSS REFERENCE
NODE FILE
NODE
NODE
FEATURE CROSS REFERENCE
FEATURE FILE
FEATURE
COORDINATE FILE
FEATURE, EDGE, AND COORDINATE FILES
EDGE AND NODE FILES

Figure 2.4

Coordinate File Layout

WORD #	BYTE #	# BYTES	TYPE	DESCRIPTION	MNEMONIC
1	1-4	4	Integer	Unique coordinate set identifier (starting at 1). A negative number indicates deleted coordinate set.	CORKEY
2	5	1	Integer	Usage edge; node; graphics.	USAGE
2	6	1	Integer	Storage: x,y; x,y,z; elipse; circle; hyperbol; spline; etc.	STORAGE
2	7-8	2	Integer	Number of coordinate pairs or triplets.	NUMCOR
3	9-12	4	Integer	ID of node, edge or graphics.	KEY
4	13-16	4		Reserved	
5-6-7 . . . n	17-n	4,8 or 12	Integer	X,Y or X,Y,Z coordinate sets for node or edge may be in compacted or full integer representation. Values are stored for graphics.	CORDATA

Repeat for next coordinate set ID.

RELATIONSHIP OF FEATURE FILE TO OTHER FILES

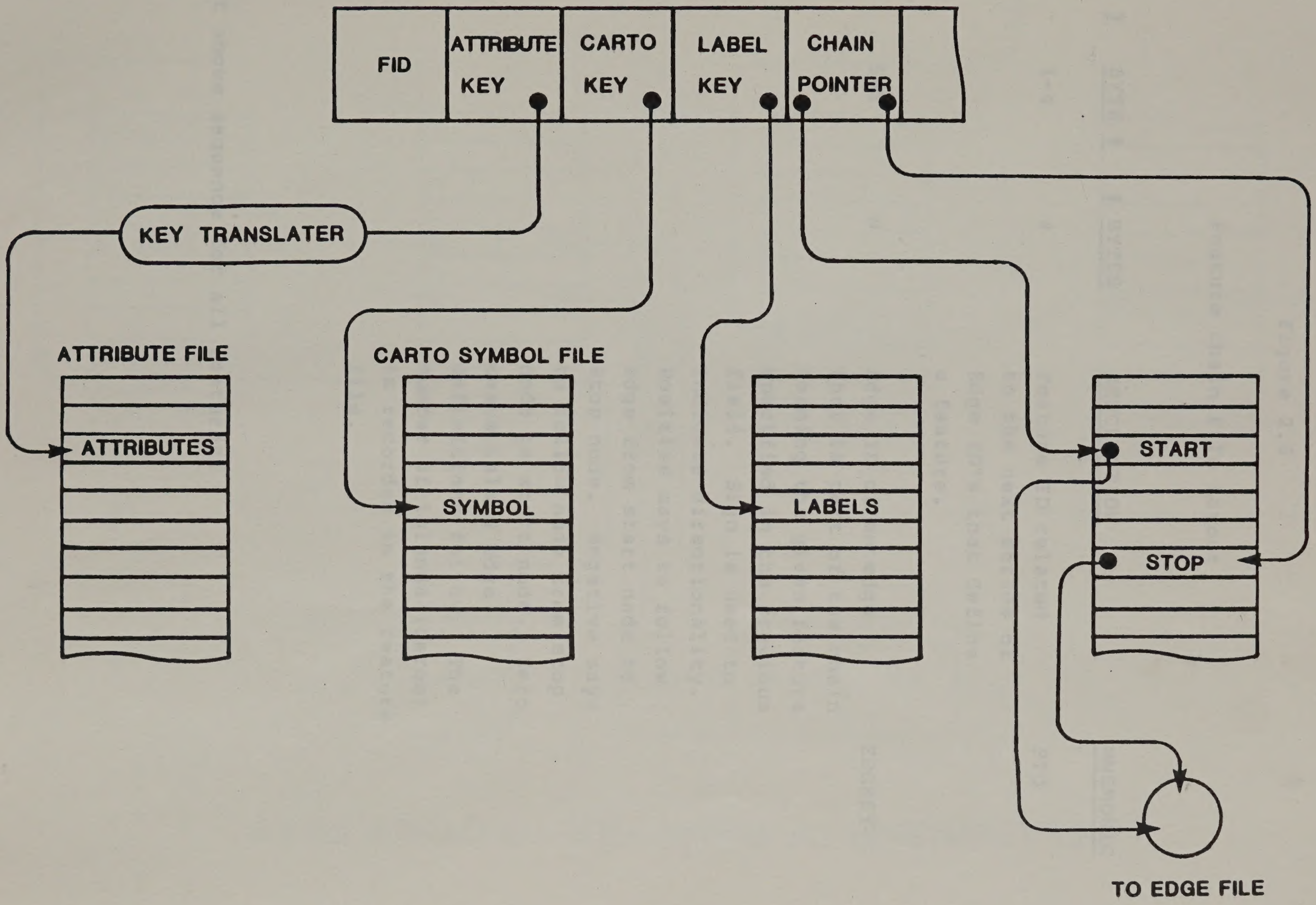

FID
ATTRIBUTE KEY
CARTO KEY
LABEL KEY
CHAIN POINTER
KEY TRANSLATER
ATTRIBUTE FILE
ATTRIBUTES
CARTO SYMBOL FILE
SYMBOL
LABELS
START
STOP
TO EDGE FILE

Figure 2.6

Feature Chain File Layout

WORD #	BYTE #	# BYTES	DESCRIPTION	MNEMONIC
1	1-4	4	Feature ID related to the next series of Edge ID's that define a feature.	FID
2-N	5-N	N	Edge ID of an edge that is part of the chain forming the given feature specified in the previous field. Sign is used to indicate directionality. Positive says to follow edge from start node to stop node. Negative says to follow node from stop node to start node. Zero means island edge definitions follow. The number of islands (zeros) is recorded in the feature file.	EDGKEY

Repeat above sequence for all features.

RELATIONSHIP OF CHAIN FILE TO OTHER FILES

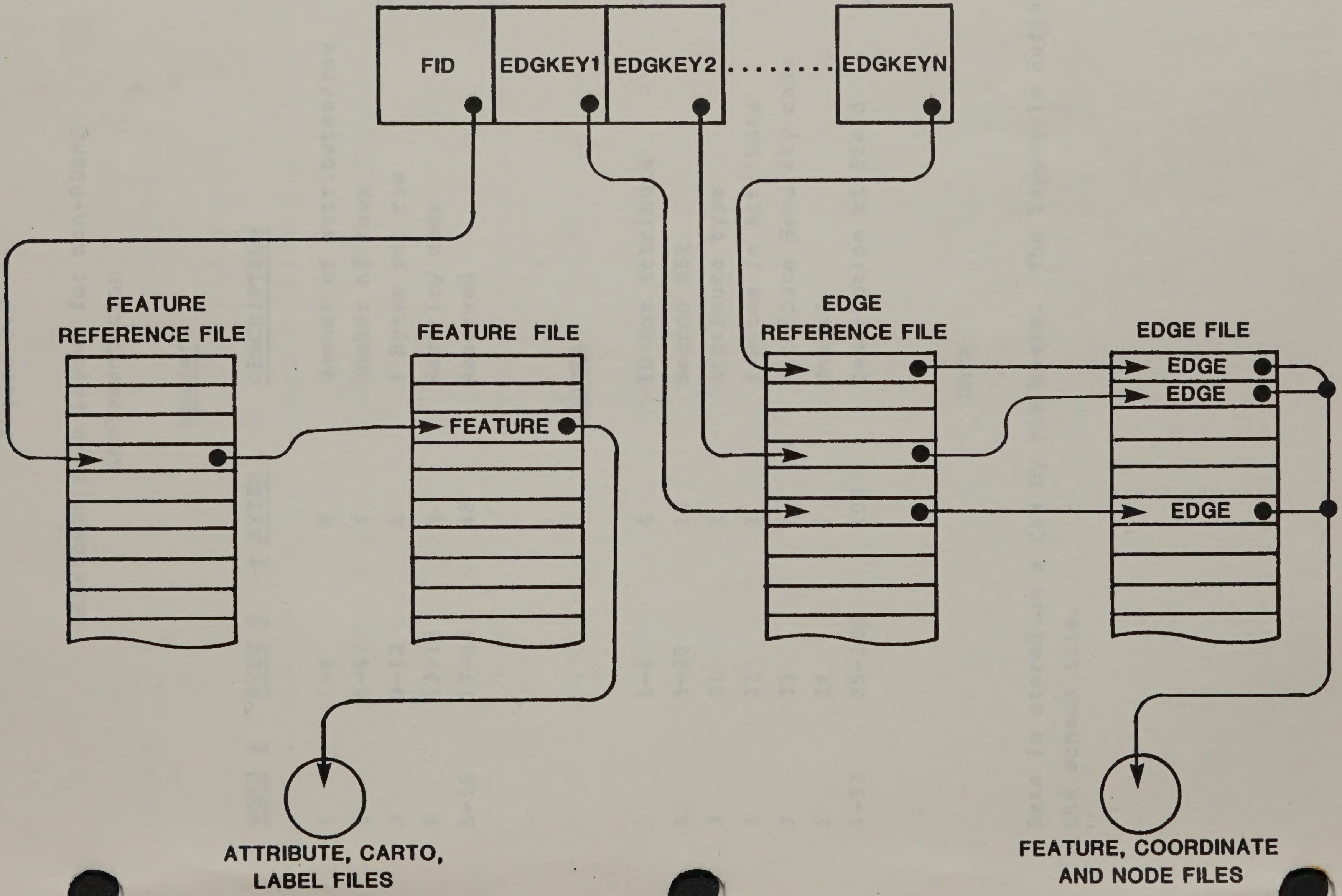

FID
EDGKEY1
EDGKEY2
EDGKEYN
FEATURE REFERENCE FILE
FEATURE FILE
FEATURE
EDGE REFERENCE FILE
EDGE FILE
EDGE
EDGE
EDGE
ATTRIBUTE, CARTO, LABEL FILES
FEATURE, COORDINATE AND NODE FILES

Figure 2.7

Attribute file layout for non-RDBMS Implementation

Header

WORD #	BYTE #	# BYTES	DESCRIPTION	MNEMONIC
1	1-4	4	Number of attributes/row	NATTS
2	5-8	4	Number of rows	NROWS
3	9-12	4	# Bytes per row	NBYTES
4	13-16	4	Security code	CODE
5-16	17-64	48	Reserved	

Schema

WORD #	BYTE #	# BYTES	DESCRIPTION	MNEMONIC
1	1-4	4	ID for attribute	ID
2	5-20	16	Search KEY	KEY
3	21	1	Attribute type	TYPE
3	22	1	# Bytes in Attribute	NBYTES
3	23	1	Attribute Security code	CODE
3	24	1	Unused	
4-32	25-128	104	Description of Field	DESCRP

DATA

Data is stored on a row by row basis. The layout is defined by the schema file.

Figure 2.9

Layout for Fast File

WORD #	BYTE #	# BYTES	TYPE	DESCRIPTION	MNEMONIC
1	1- 4	4	INTEGER	Minimum Y value	XMIN
2	5- 8	4	INTEGER	Minimum X value	YMIN
3	9-12	4	INTEGER	Minimum Z value	ZMIN
4	13-16	4	INTEGER	Maximum X value	XMAX
5	17-20	4	INTEGER	Maximum Y value	YMAX
6	21-24	4	INTEGER	Maximum Z value	ZMAX
7	25-28	4	INTEGER	Feature ID	FP
8	29-32	4	INTEGER	Attribute ID	AP

Figure 2.10

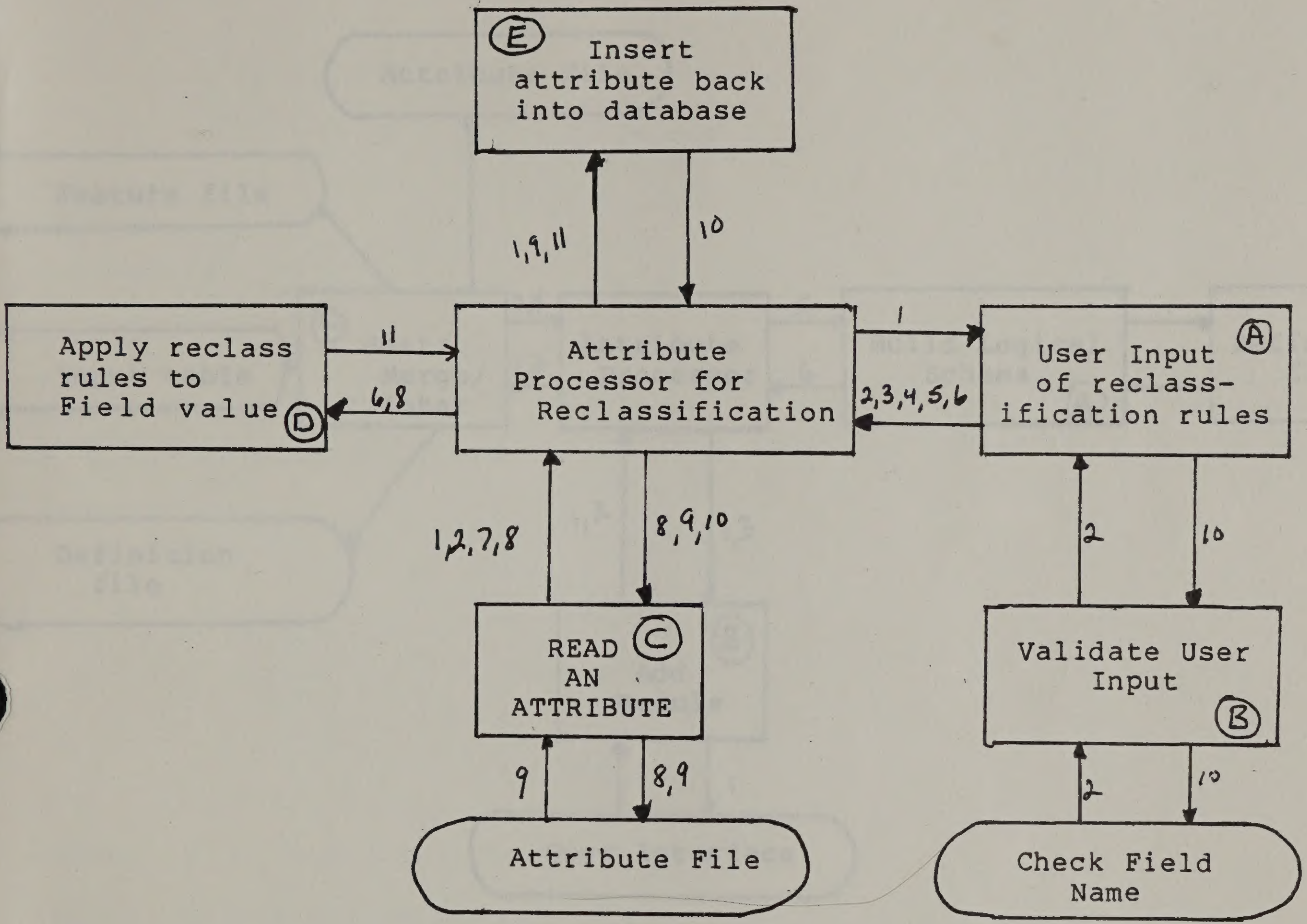
FILE NAME: WORK AND MASTER HEADERS
RECORD TYPE: .DH FILE, .HD OF CELL FILES, AND FIRST TWO RECORDS OF VECTOR FILES

WORD #	BYTE #	# BYTES	TYPE	DESCRIPTION
1	1-2	2	I	NUMBER OF 128 WORD RECORDS IN FILE
2-11	3-22	20	CHAR	NAME OF DIGITIZING SYSTEM
12-15	23-30	8	CHAR	DATE OF ENTRY TO MOSS
16-19	31-38	8	CHAR	STUDY AREA NAME
20-49	39-98	60	CHAR	MAP DESCRIPTION
50	99-100	2		RESERVED
51-52	101-104	4	CHAR	YEAR OF SOURCE MAP
52-57	105--114	10	CHAR	COORDINATE PROJECTION DESCRIPTION
58-59	115-118	4		RESERVED
60	119-120	2	INTE	# OF ROWS-CELL MAPS (UNUSED FOR VECTOR)
61	121-122	2	INT	# OF COLUMNS-CELL MAPS (UNUSED FOR VECTOR
62	123-124	2	INTE	TYPE 1-POINT 2-LINE 3-AREA 4-ARC/NODE 5-ELEV PT 6-BIT 7-DISCRETE 8-CONTINUOUS 10-TEXT 11-3D POINT 12-3D LINE 13-3D POLYGON
63	125-126	2	INTE	# OF FEATURES (UNUSED FOR CELL)
64	127-128	2		RESERVED (VALUE IS ONE)
65	129-130	2	INTE	ESTIMATED # OF UNIQUE ATTRIBUTES (UNUSED FOR CELL)
66	131-132	2		RESERVED (VALUE IS ZERO)
67	133-134	2	INTE	ACTUAL # OF SUBJECTS
68-69	135-138	4	REAL	MINIMUM X OF MBR (WEST)
70-71	139-142	4	REAL	MAXIMUM X OF MBR (EAST)
72-73	143-146	4	REAL	MINIMUM Y OF MBR (SOUTH)

74-75	147-150	4	REAL	MAXIMUM Y OF MBR (NORTH)
76	151-152	2	INTE	MULTIPLE ATTRIBUTES FILE (.AT) PRESENCE 0-NO 1-YES. FOR VECTOR FILES ONLY.
77	153-154	2	CHAR	UNITS OF VECTOR MAP F-FEET M-METERS
78-79	155-158	4	REAL	ACRES PER CELL
80-81	159-162	4	REAL	HEIGHT OF CELL IN METERS
82-83	163-166	4	REAL	WIDTH OF CELL IN METERS
84-85	167-170	4	REAL	NUMBER OF SET BINS TYPE 6 (DICHOTOMOUS)
86-87	171-174	4		RESERVED
88-89	175-178	4	REAL	MINIMUM CELL VALUE TYPE 8 (CONTINUOUS)
90-91	179-182	4	REAL	MAXIMUM CELL VALUE TYPE 8 (CONTINUOUS)
92-119	183-238	56		RESERVED
120	239-240	2	INTE	FONT # ZERO IF NOT FONT ASSIGNED (UNUSED FOR CELL)
121-168	241-336	96		RESERVED
169-176	337-352	16	INTE	POINTER & LAST 7 COMMAND NUMBERS
177-192	353-384	32		SAME AS 16 WORDS OF .DT RECORD FOR MAP
193-256	385-512	128		PROJECTION INFORMATION (-1 IF NONE ASSIGNED) 193-PROJECTION NUMBER (0-20) 194-ZONE 195-UNITS 0-RADIANS 1-FEET 2-METERS 3-SECONDS 4-DEGREES 196-ELLIPSE TYPE (0-19) 197-256 15 DOUBLE PRECISION PARAMETERS

Figure 3.1

Attribute Reclassification

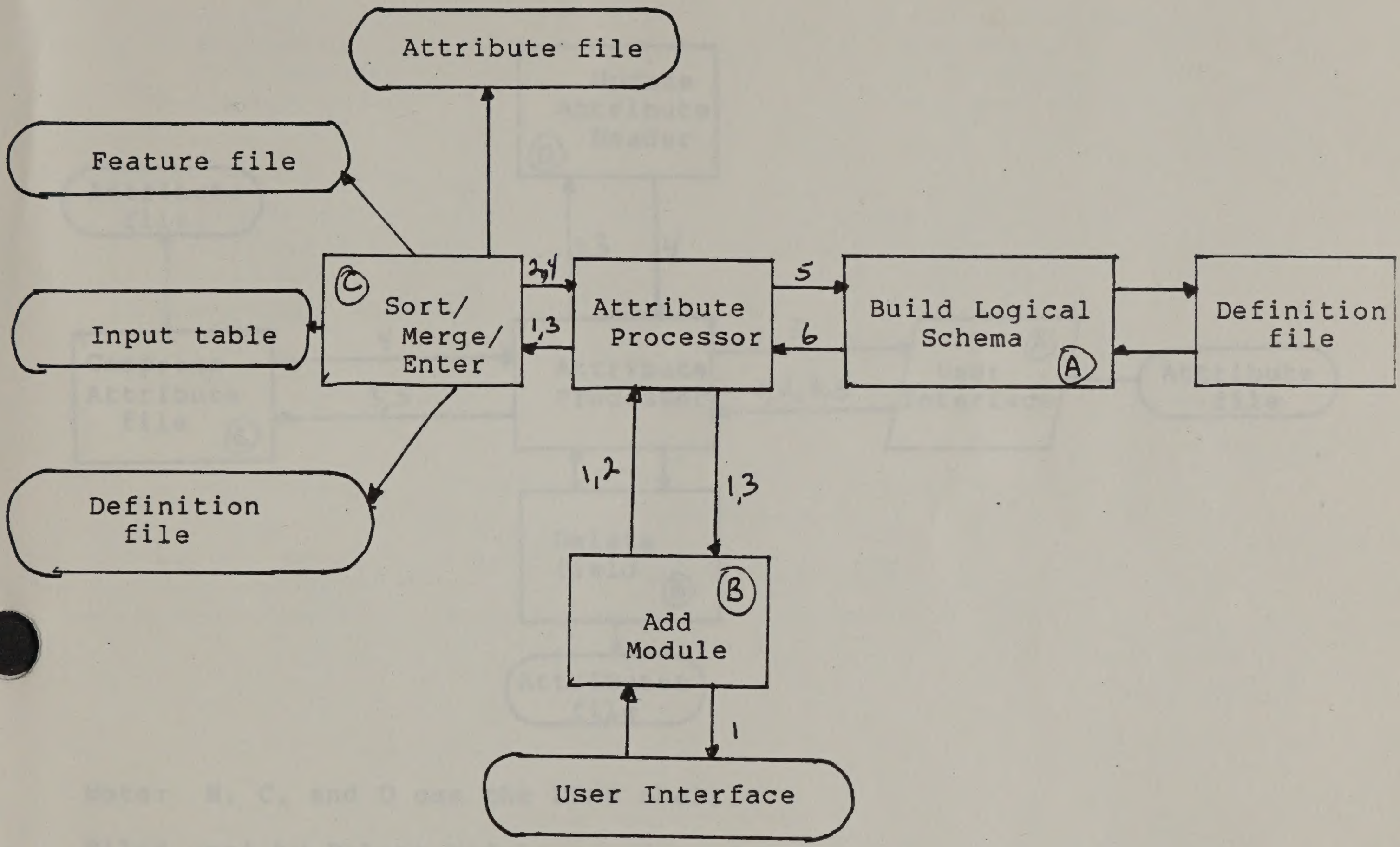


Ⓔ Insert attribute back into database
1,9,11
10
Apply reclass rules to Field value Ⓓ
11
6,8
Attribute Processor for Reclassification
1
2,3,4,5,6
User Input of reclass-ification rules Ⓐ
1,2,7,8
8,9,10
2
10
READ AN ATTRIBUTE Ⓒ
Validate User Input Ⓑ
9
8,9
2
10
Attribute File
Check Field Name

Note: B, C, and E are part of the DBMS shell.

Figure 3.2

Attribute Entry + Validation


Attribute file
Feature file
Input table
Definition file
Ⓒ
Sort/
Merge/
Enter
2,4
1,3
Attribute
Processor
5
6
Build Logical
Schema
Ⓐ
Definition
file
1,2
1,3
Ⓑ
Add
Module
1
User Interface

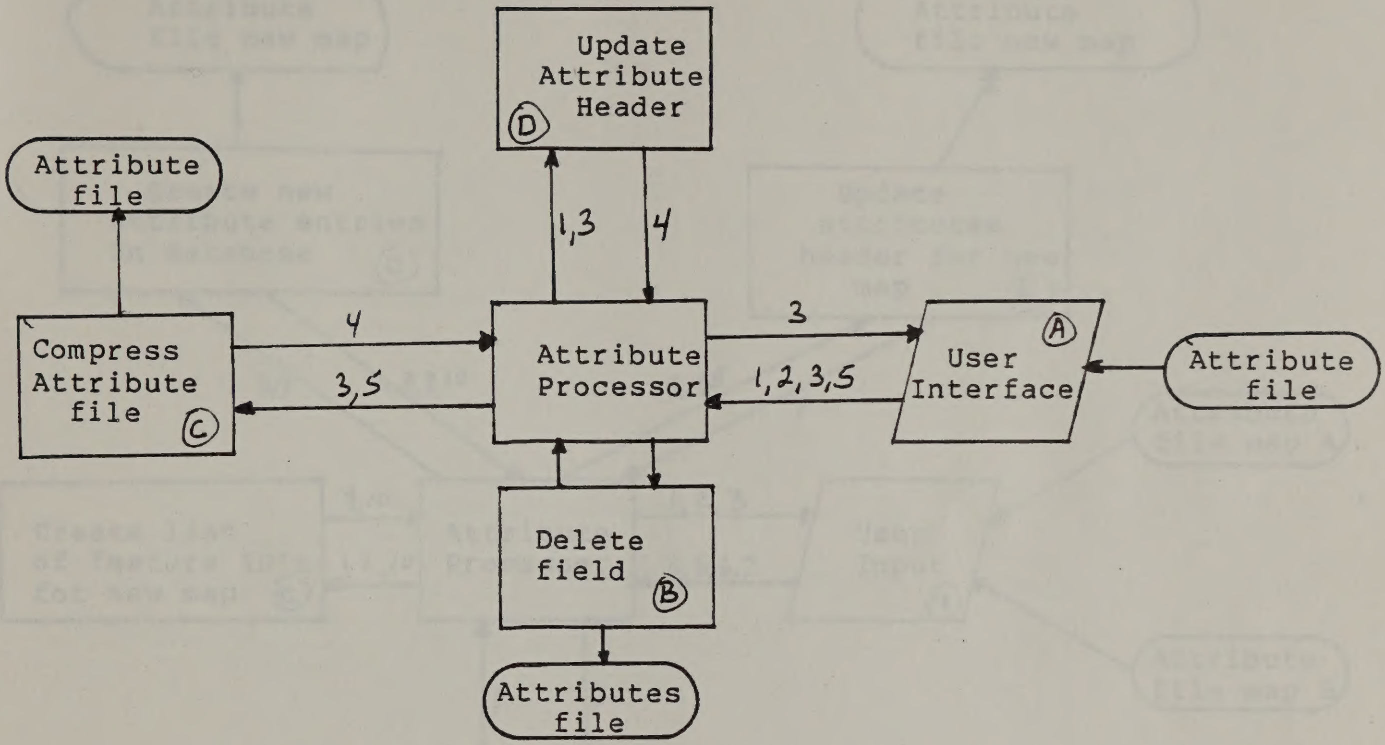
Note: The Sort/merge module uses the following files:

1. Feature file
2. Attribute file
3. A logical schema definition file that is created using the schema create and edit module.
4. An Input Table of values to be loaded into the attributes database. This is a matrix of values in row column format.

Figure 3.4

Attribute Delete

Attribute file


Update
Attribute
Header
D
1,3
4
Attribute
file
Compress
Attribute
file
C
4
3,5
Attribute
Processor
3
1,2,3,5
User
Interface
A
Attribute
file
Delete
field
B
Attributes
file

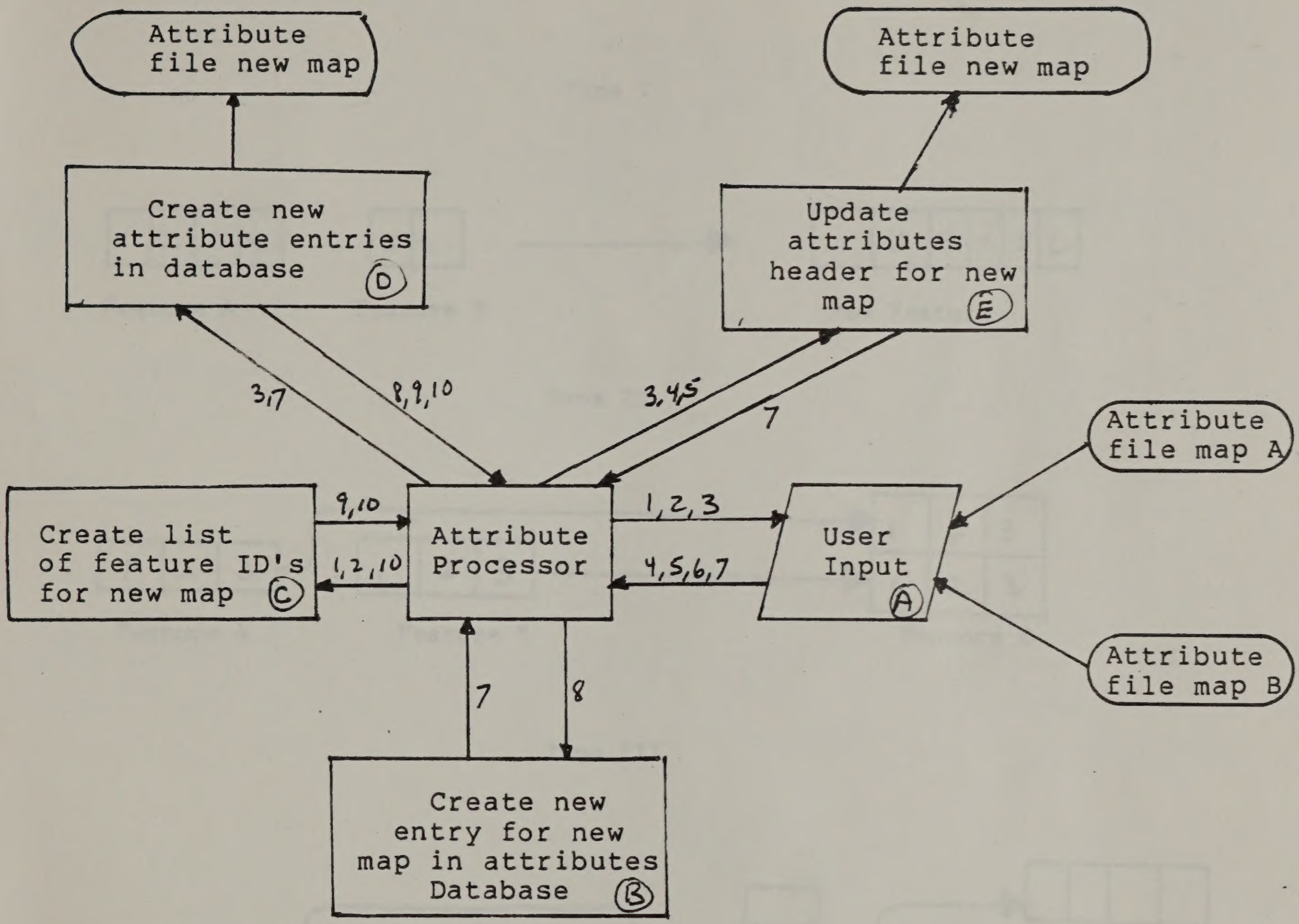
Note: B, C, and D use the DBMS shell.

Files used by Delete Module

1. Attribute file

Figure 3.5

Attribute Merge: Type 1


Attribute file new map
Create new attribute entries in database
D
Attribute file new map
Update attributes header for new map
E
3,7
8,9,10
3,4,5
7
Attribute file map A
Create list of feature ID's for new map
C
9,10
1,2,10
Attribute Processor
1,2,3
4,5,6,7
User Input
A
Attribute file map B
7
8
Create new entry for new map in attributes Database
B

Note: A, C, D, and E use the DBMS shell

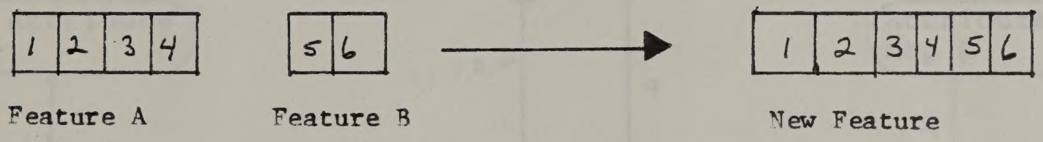
Files used

1. Attribute file for map A
2. Attribute file for map B

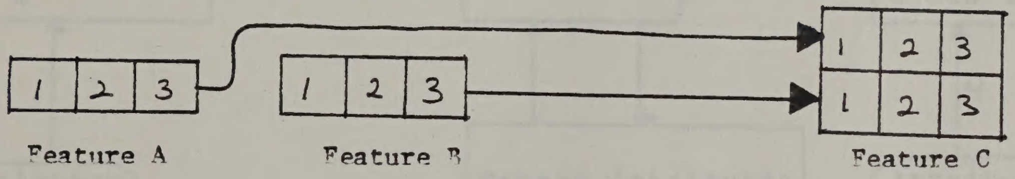
Figure 3.5

Attribute Merge

Type I


1 2 3 4
Feature A
5 6
Feature B
1 2 3 4 5 6
New Feature

Type II


1 2 3
Feature A
1 2 3
Feature B
1 2 3
1 2 3
Feature C

Type III

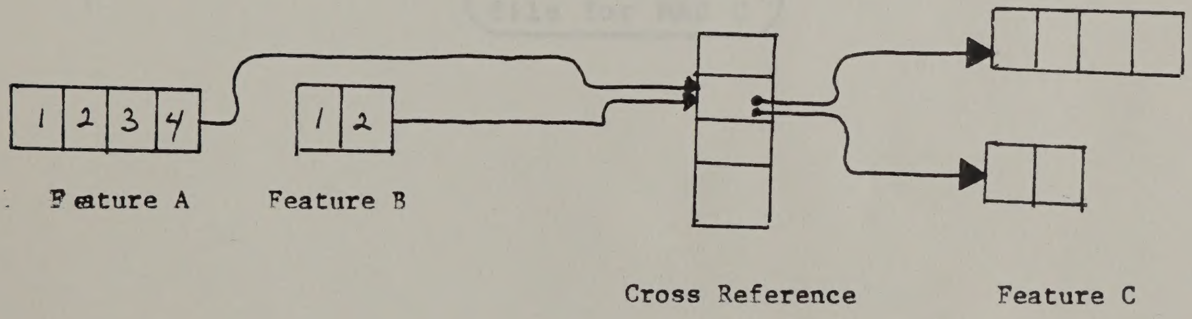
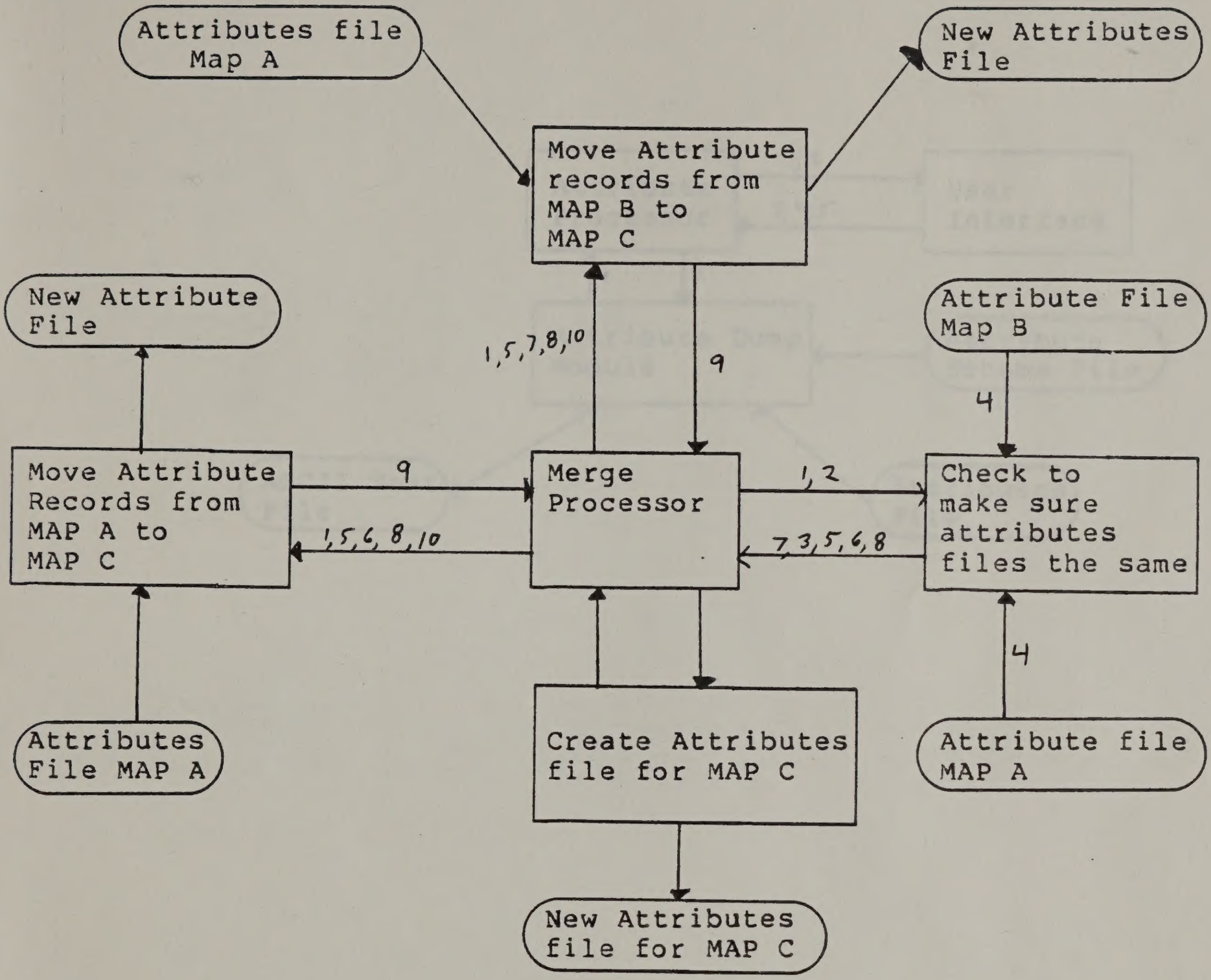

1 2 3 4
Feature A
1 2
Feature B
Cross Reference
Feature C

Figure 3.5.1

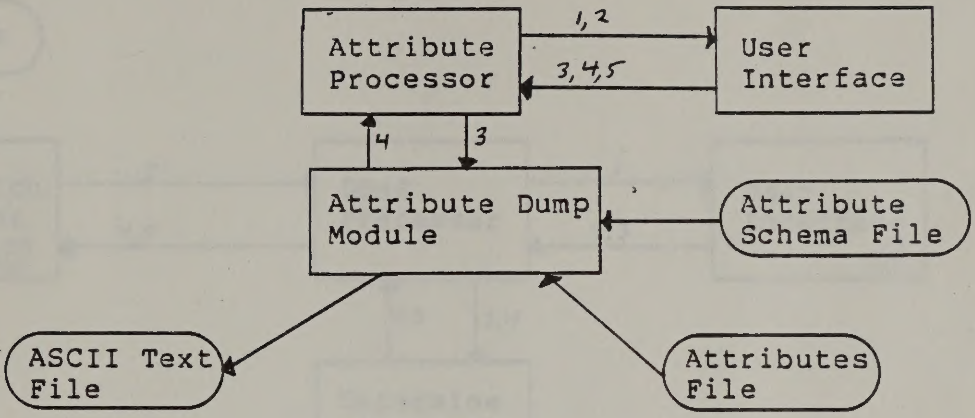
Attribute Merge: Type II


Attributes file Map A
New Attributes File
Move Attribute records from MAP B to MAP C
New Attribute File
1,5,7,8,10
9
Attribute File Map B
4
Move Attribute Records from MAP A to MAP C
9
Merge Processor
1,2
Check to make sure attributes files the same
1,5,6,8,10
7,3,5,6,8
4
Attributes File MAP A
Create Attributes file for MAP C
Attribute file MAP A
New Attributes file for MAP C

File Used
1) Attributes file input map A
2) Attributes file input map B
3) Attribute file output map C

Figure 3.6

Attribute Dump to Interchange File

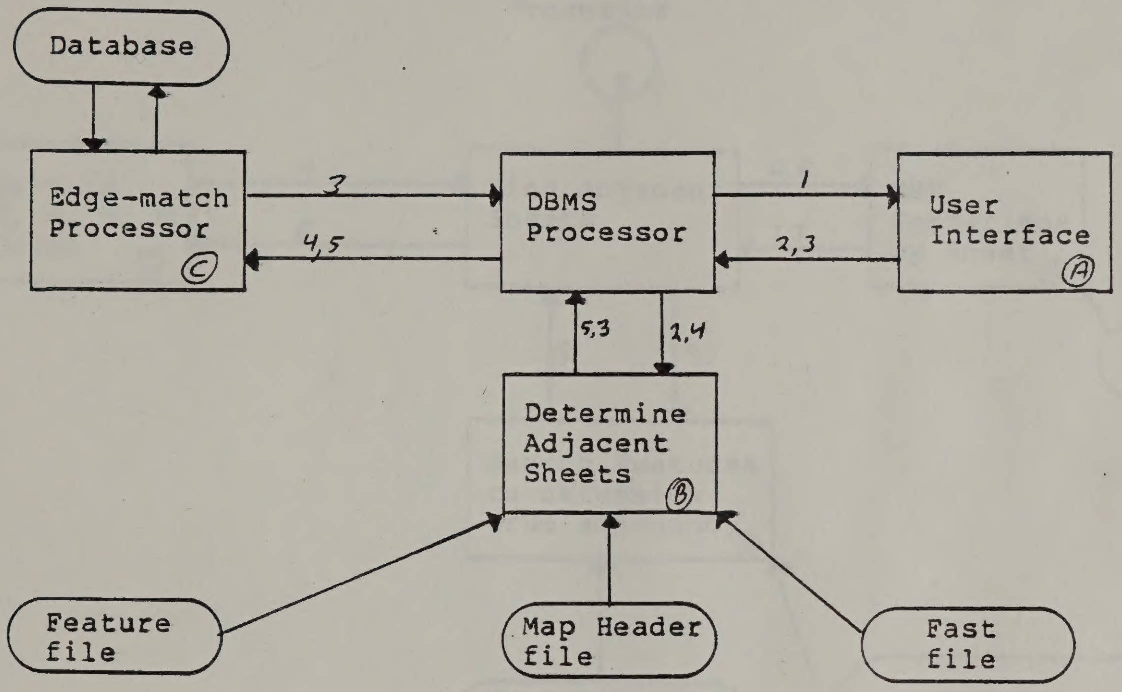

Attribute Processor
1,2
User Interface
3,4,5
4
3
Attribute Dump Module
Attribute Schema File
ASCII Text File
Attributes File

Files Used

1) Attribute Schema File
2) Map Attributes File
3) ASCII text file for output attributes exchange file

Figure 3.7

Edgematch Process

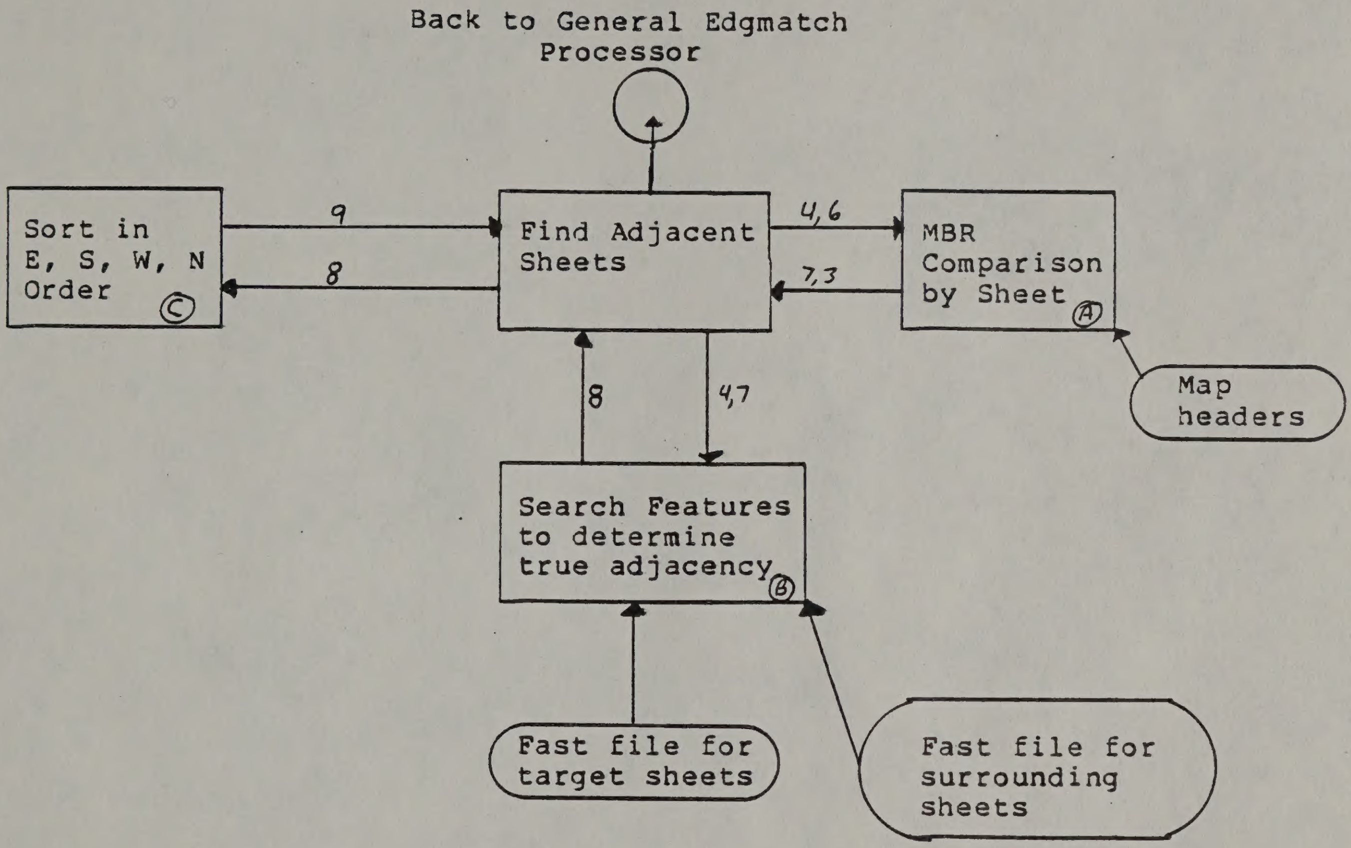

Database
Edge-match Processor
C
3
4,5
DBMS Processor
1
2,3
User Interface
A
5,3
2,4
Determine Adjacent Sheets
B
Feature file
Map Header file
Fast file

Files Used:

1) Fast index file
2) Map header file
3) Database file - uses edge, feature, and coordinate files for both input maps

Figure 3.8

Determine Adjacent Sheets

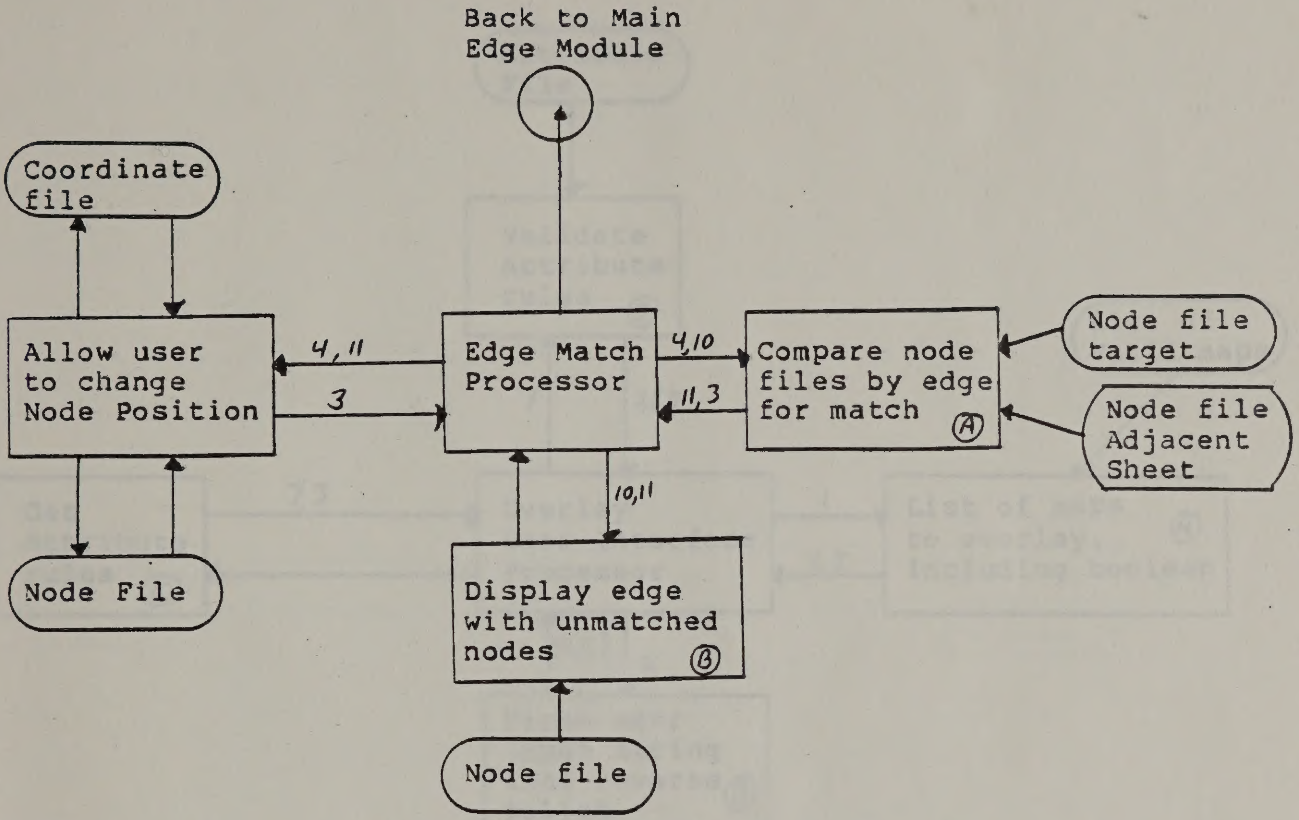

Back to General Edgmatch Processor
Sort in E, S, W, N Order
Ⓒ
9
8
Find Adjacent Sheets
4,6
7,3
MBR Comparison by Sheet
Ⓐ
Map headers
8
4,7
Search Features to determine true adjacency
Ⓑ
Fast file for target sheets
Fast file for surrounding sheets

Files Used:

1) Map header file
2) Fast index file for target and 4 surrounding sheets
3) Feature file

Figure 3.9

Edge Match Processor



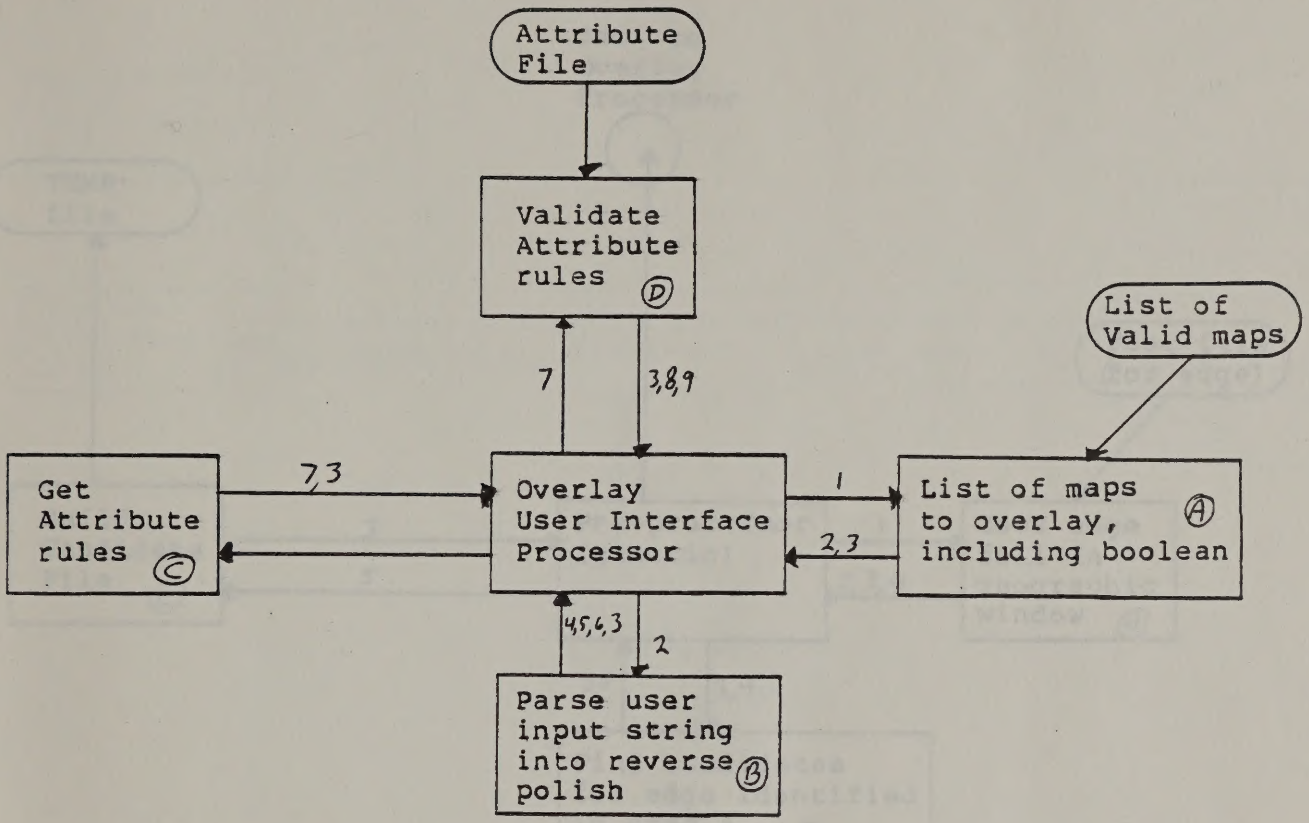
Back to Main Edge Module
Coordinate file
Allow user to change Node Position
4,11
3
Edge Match Processor
4,10
11,3
Compare node files by edge for match
Ⓐ
Node file target
Node file Adjacent Sheet
10,11
Node File
Display edge with unmatched nodes
Ⓑ
Node file

Files Used:

1) Node file for target map sheet
2) Node files for four adjacent sheets
3) Coordinate file
4) Edge file

Figure 3.10

User Interfact to Overlay Process


Attribute File
Validate Attribute rules Ⓓ
List of Valid maps
7
3,8,9
Get Attribute rules Ⓒ
7,3
Overlay User Interface Processor
1
2,3
List of maps to overlay, including boolean Ⓐ
4,5,6,3
2
Parse user input string into reverse polish Ⓑ

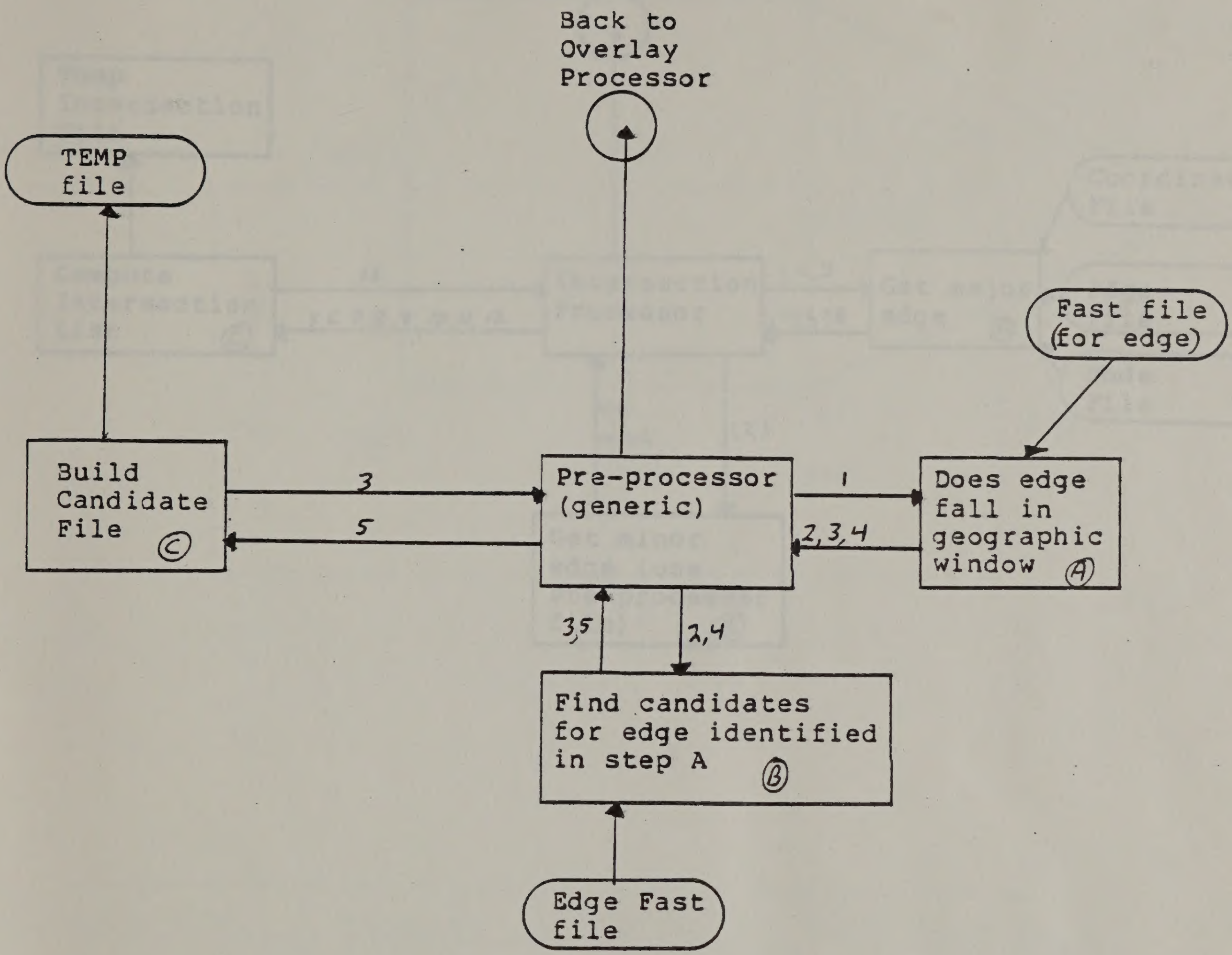
B. This software exists - RPN.
D. This software exists - POVO.

Files Used:

1) Map names file
2) Attribute file of input maps

Figure 3.11

Spatial Pre-processor to determine candidates


Back to
Overlay
Processor
TEMP file
Build Candidate File Ⓒ
3
5
Pre-processor (generic)
1
2,3,4
Does edge fall in geographic window Ⓐ
Fast file (for edge)
3,5
2,4
Find candidates for edge identified in step A Ⓑ
Edge Fast file

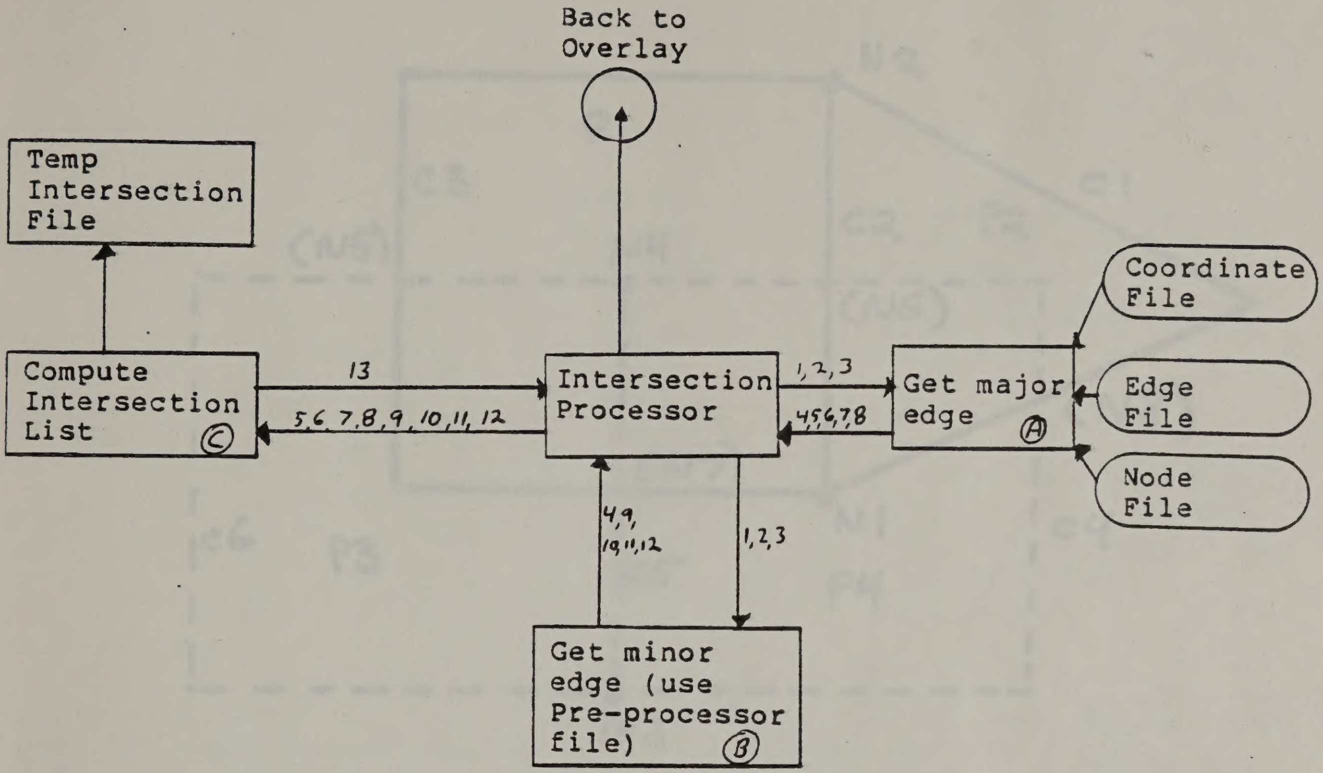
* This exists in form of subroutine called PREPRO

Files Used:

1) Fast Index files for input maps
2) A temp file that stores candidate EDGE ID's

Figure 3.12

Intersection Processor

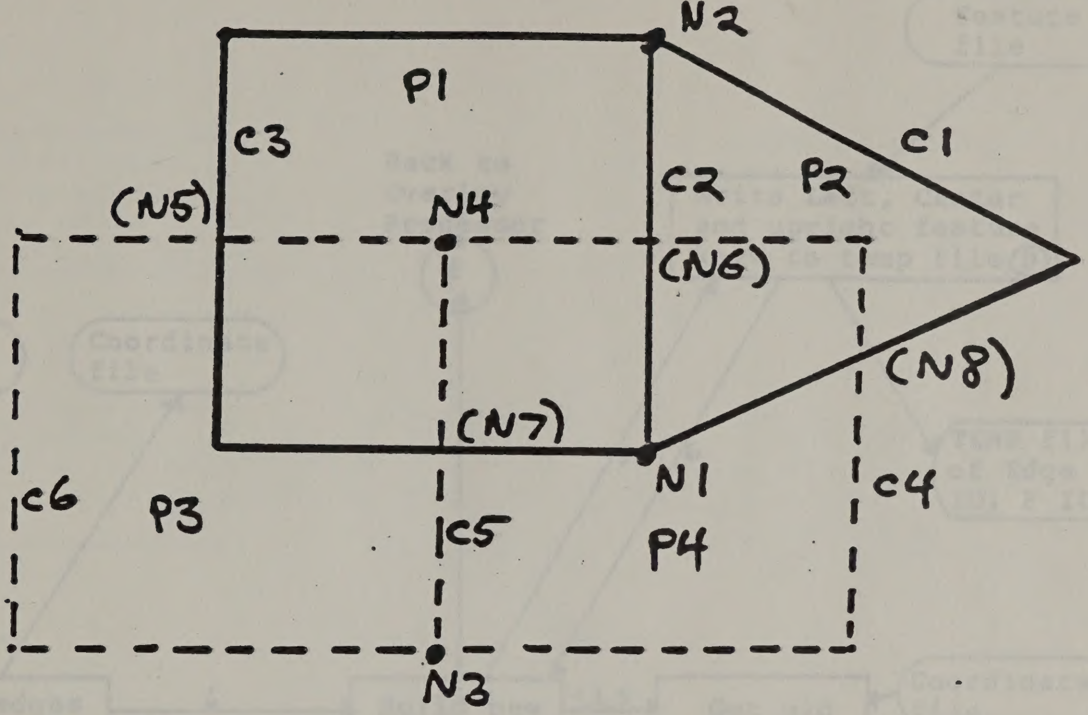

Back to Overlay
Temp Intersection File
Compute Intersection List ©
13
5,6,7,8,9,10,11,12
Intersection Processor
1,2,3
4,5,6,7,8
Get major edge Ⓐ
Coordinate File
Edge File
Node File
4,9,
10,11,12
1,2,3
Get minor edge (use Pre-processor file) Ⓑ

Files Used:

1) Coordinate file
2) Edge file
3) Node file
4) TEMP file created by spatial pre-procesor
5) A new TEMP file that stores intersection lists

Figure 3.13

Overlaying Map A (solid lines) on Map B (dashed lines)

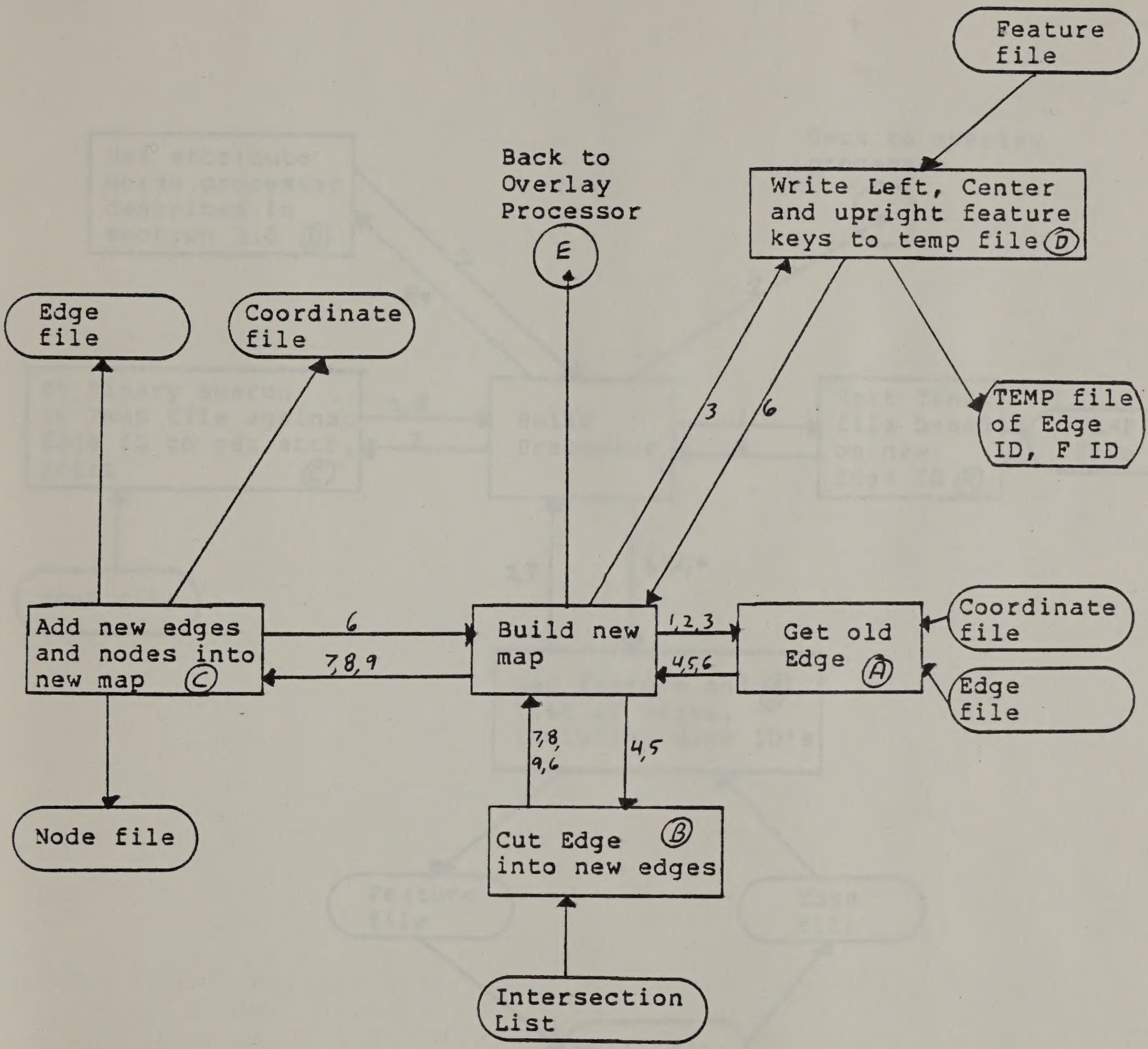

N2
P1
C3
C1
C2
P2
(N5)
N4
(N6)
(N8)
(N7)
N1
C4
C6
P3
C5
P4
N3

Chain No.	Edge No.	Node No.	Other Chain No.
3	2	5	6
6	1	5	3
2	1	6	4
4	3	6	2
5	1	7	3
3	1	7	5
1	1	8	4
4	2	8	1

Intersection Data Structure

Figure 3.14

Build New Arc/Node Map from Intersection List

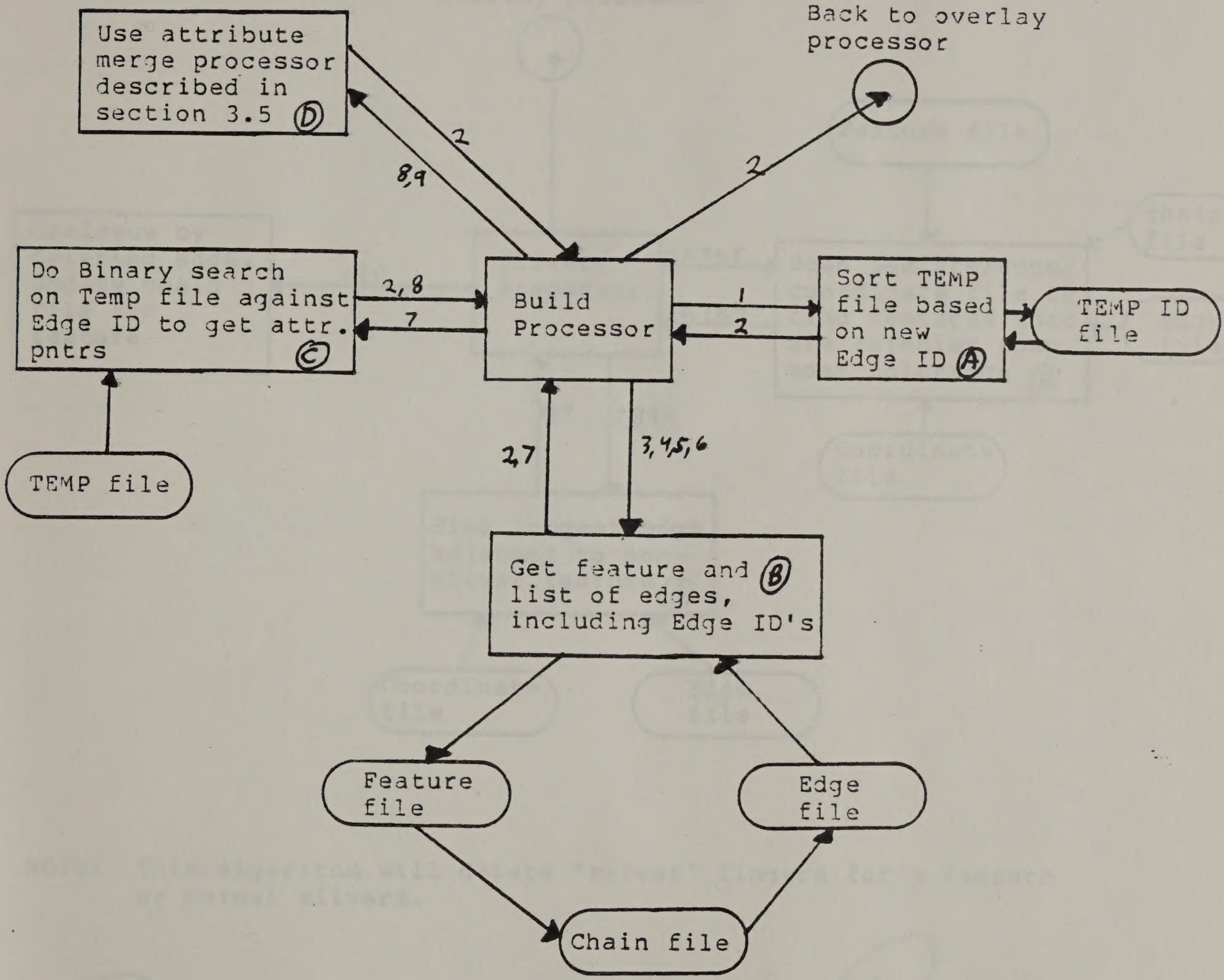

Feature file
Back to Overlay Processor
Write Left, Center and upright feature keys to temp file (D)
E
Edge file
Coordinate file
3
6
TEMP file of Edge ID, F ID
Add new edges and nodes into new map (C)
6
7,8,9
Build new map
1,2,3
4,5,6
Get old Edge (A)
Coordinate file
Edge file
7,8, 9,6
4,5
Node file
Cut Edge into new edges (B)
Intersection List

Files Used:

1) Coordinate file from old maps
2) Edge file from old maps
3) Intersection list file
4) Node file for new map
5) Edge file for new map
6) Coordinate file for new map

Figure 3.15

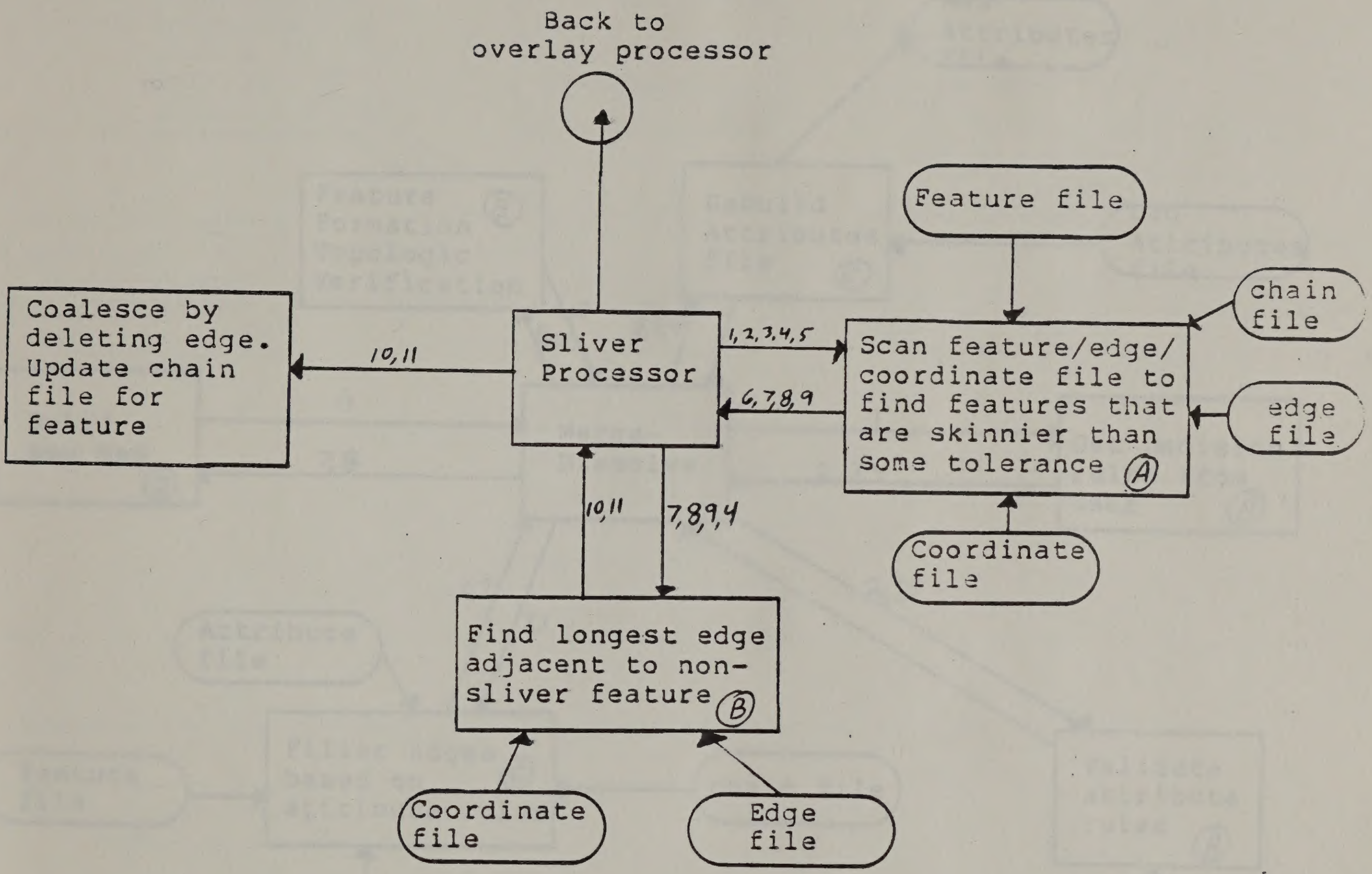
Attribute Build Processor



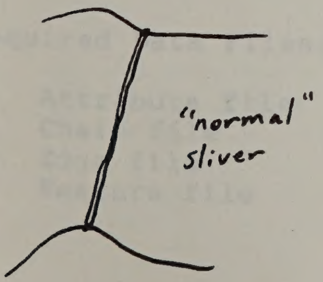
Use attribute merge processor described in section 3.5 Ⓓ
Back to overlay processor
2
8,9
2
Do Binary search on Temp file against Edge ID to get attr. pntrs Ⓒ
2,8
7
Build Processor
1
2
Sort TEMP file based on new Edge ID Ⓐ
TEMP ID file
TEMP file
2,7
3,4,5,6
Get feature and list of edges, including Edge ID's Ⓑ
Feature file
Edge file
Chain file

Figure 3.16

Sliver Removal


Back to overlay processor
Coalesce by deleting edge. Update chain file for feature
10,11
Sliver Processor
1,2,3,4,5
6,7,8,9
Feature file
Scan feature/edge/ coordinate file to find features that are skinnier than some tolerance Ⓐ
chain file
edge file
Coordinate file
10,11
7,8,9,4
Find longest edge adjacent to non-sliver feature Ⓑ
Coordinate file
Edge file

NOTE: This algorithm will delete 'sliver' fingers for a feature or normal slivers.


"normal" sliver

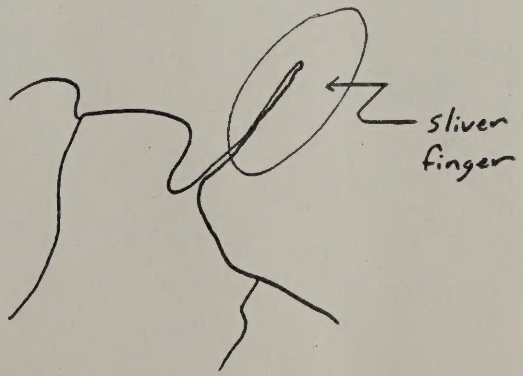
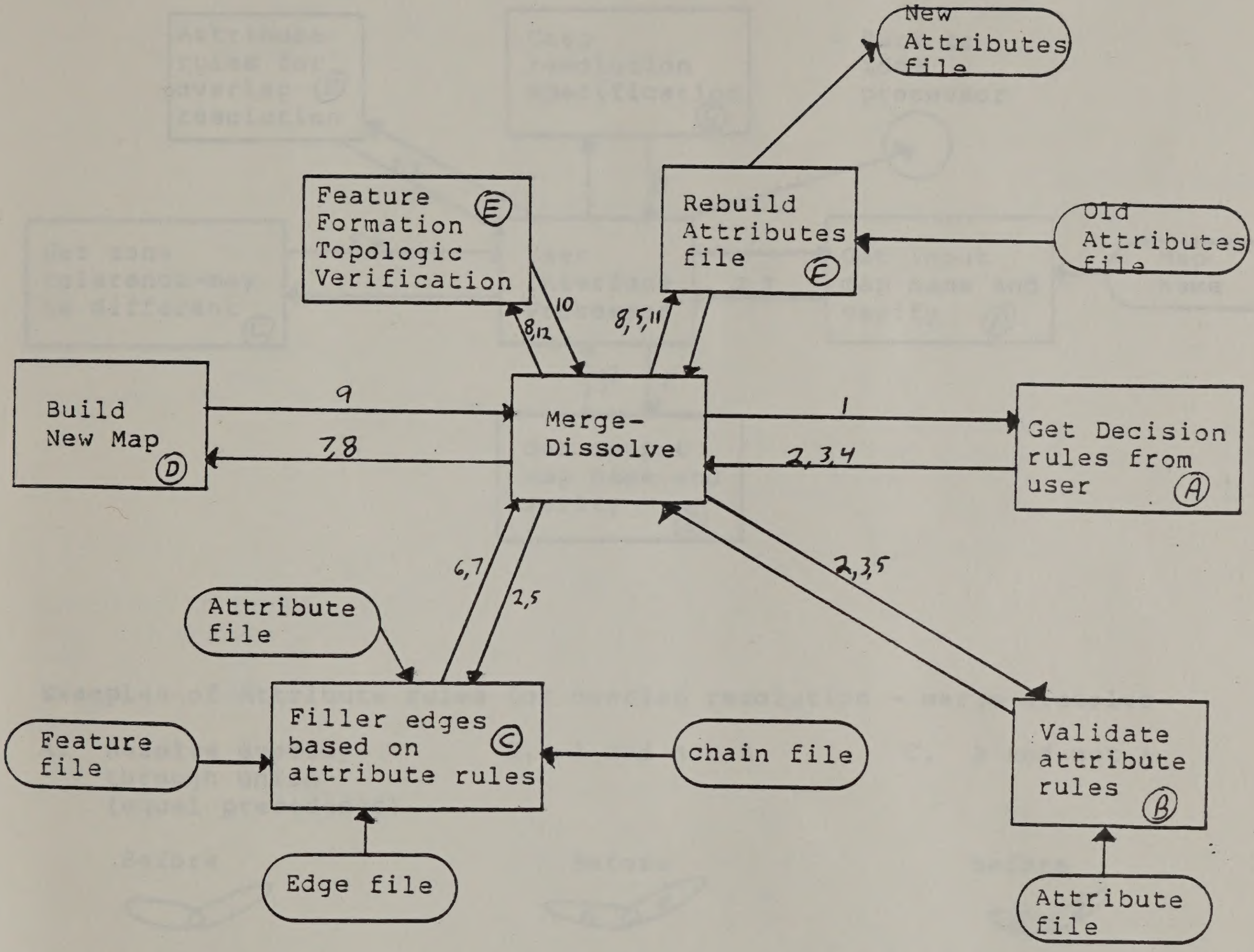

sliver finger

Figure 3.17

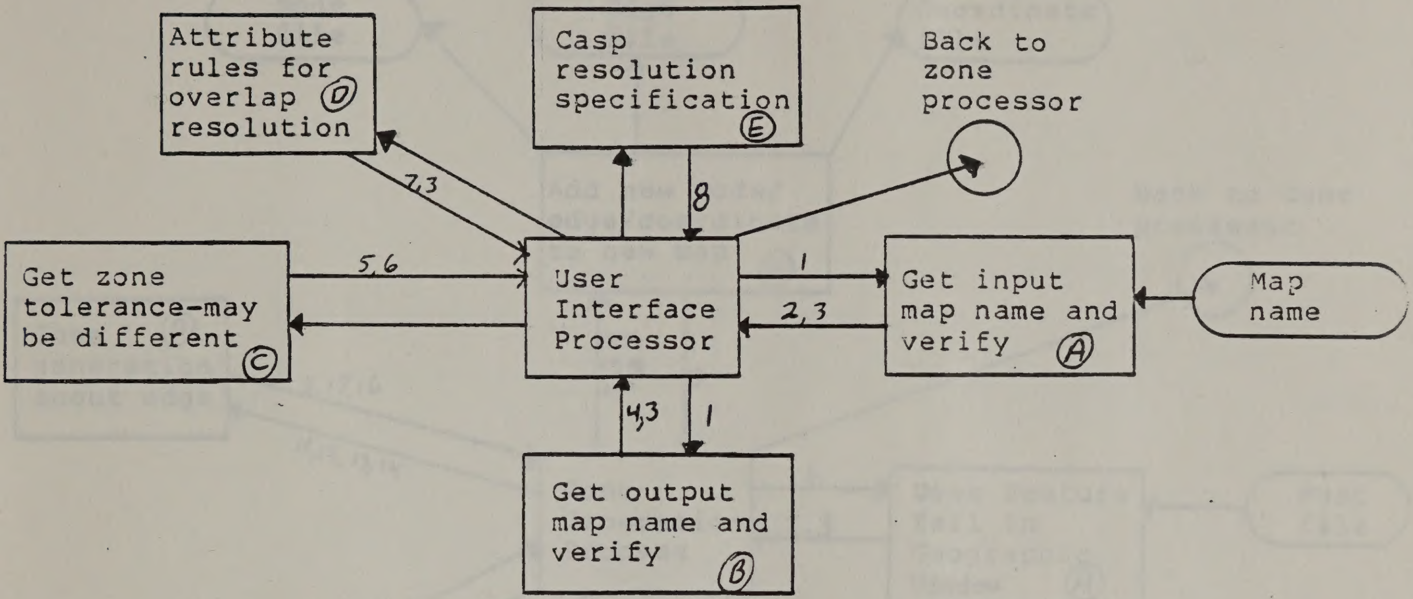

New Attributes file
Feature Formation Topologic Verification Ⓔ
Rebuild Attributes file Ⓔ
Old Attributes file
10
8,12
8,5,11
Build New Map Ⓓ
9
7,8
Merge-Dissolve
1
2,3,4
Get Decision rules from user Ⓐ
6,7
2,5
2,3,5
Attribute file
Filler edges based on attribute rules Ⓒ
Feature file
chain file
Edge file
Validate attribute rules Ⓑ
Attribute file

Required Data Files:

1) Attribute file
2) Chain file
3) Edge file
4) Feature file

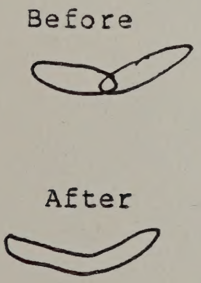
Figure 3.18

Zone Generation User Interface

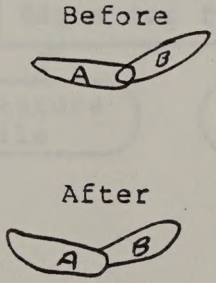

Attribute rules for overlap resolution Ⓓ
Casp resolution specification Ⓔ
Back to zone processor
2,3
8
Get zone tolerance-may be different Ⓒ
5,6
User Interface Processor
1
2,3
Get input map name and verify Ⓐ
Map name
4,3
1
Get output map name and verify Ⓑ

Examples of Attribute rules for overlap resolution - merge dissolve

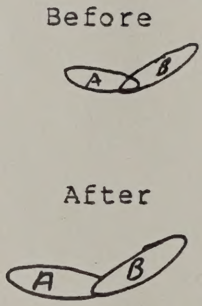
A. Resolve overlay through union (equal precedence)


Before
After

B. A and not B


Before
A
B
After
A
B

C. B and not A


Before
A
B
After
A
B

D. No precedence
(A not B and B not A)

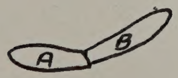


Figure 3.19

Zone Generation

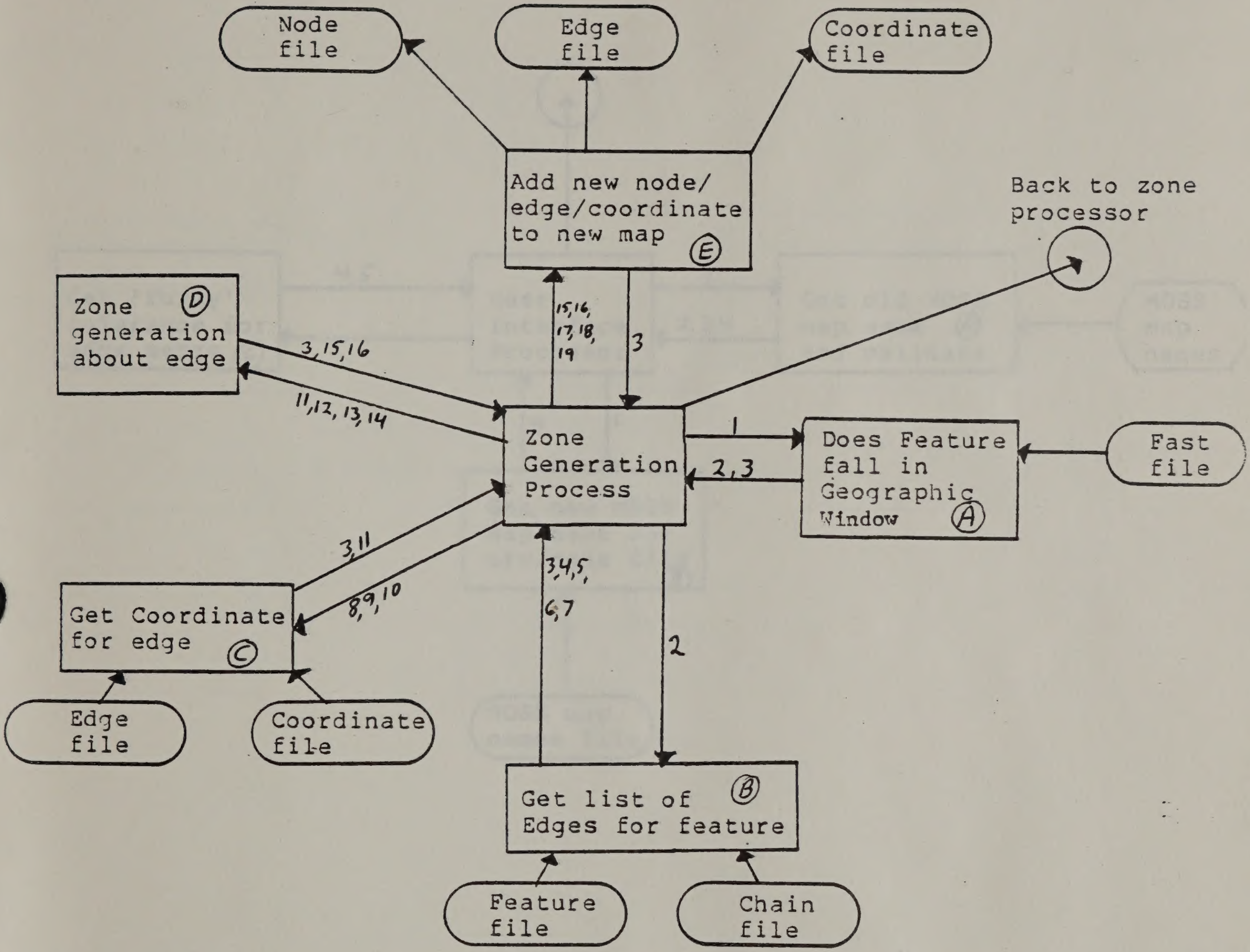
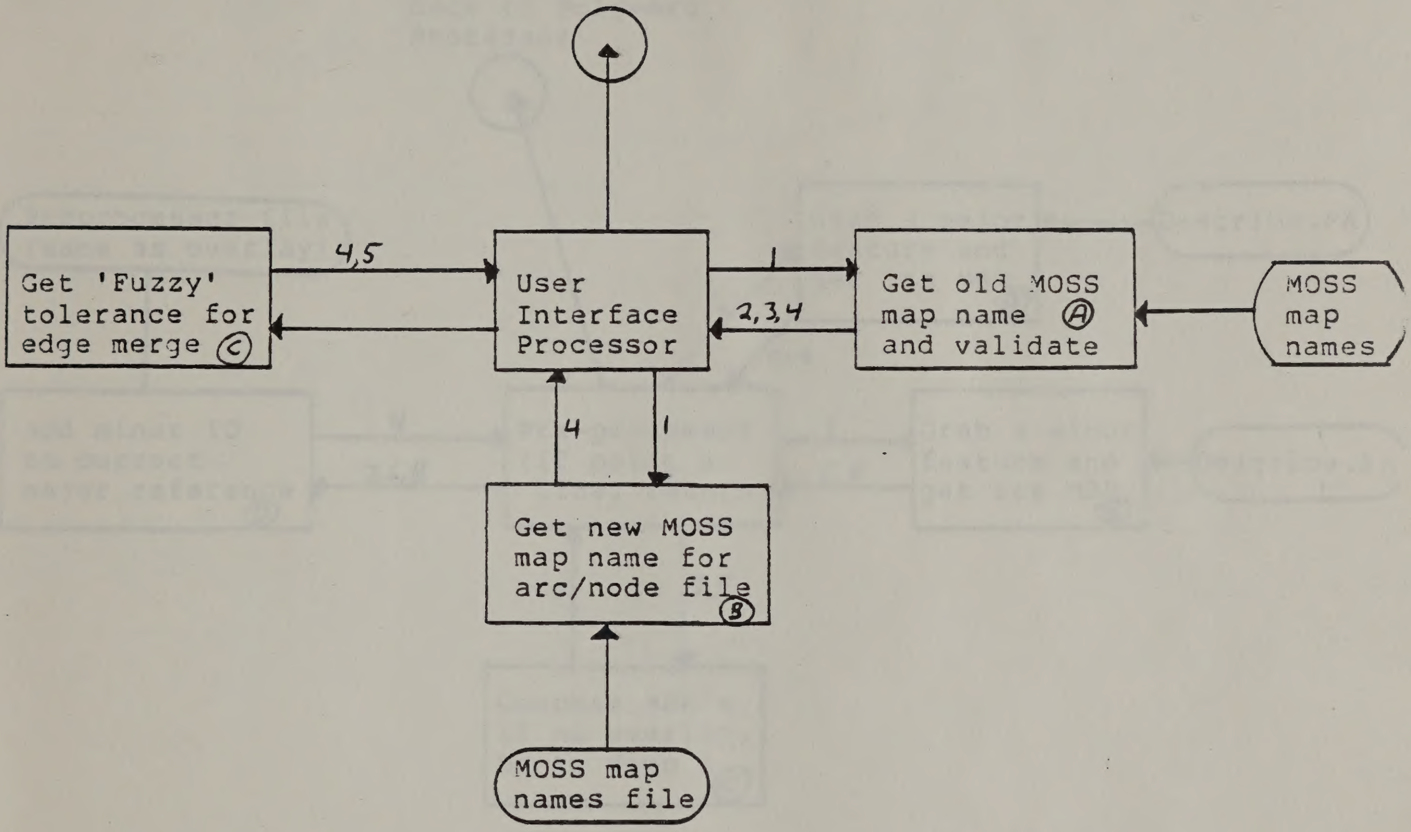

Node file
Edge file
Coordinate file
Add new node/ edge/coordinate to new map Ⓔ
Back to zone processor
Zone generation about edge Ⓓ
15,16, 17,18, 19
3
3,15,16
11,12,13,14
Zone Generation Process
1
2,3
Does Feature fall in Geographic Window Ⓐ
Fast file
3,11
8,9,10
3,4,5, 6,7
2
Get Coordinate for edge Ⓒ
Edge file
Coordinate file
Get list of Edges for feature Ⓑ
Feature file
Chain file

Figure 3.20

Polygon to Arc User Interface



Get 'Fuzzy' tolerance for edge merge ©
4,5
User Interface Processor
1
2,3,4
Get old MOSS map name and validate Ⓐ
MOSS map names
4
1
Get new MOSS map name for arc/node file Ⓑ
MOSS map names file

Figure 3.21

Pre-processor to find candidates for Poly to Arc

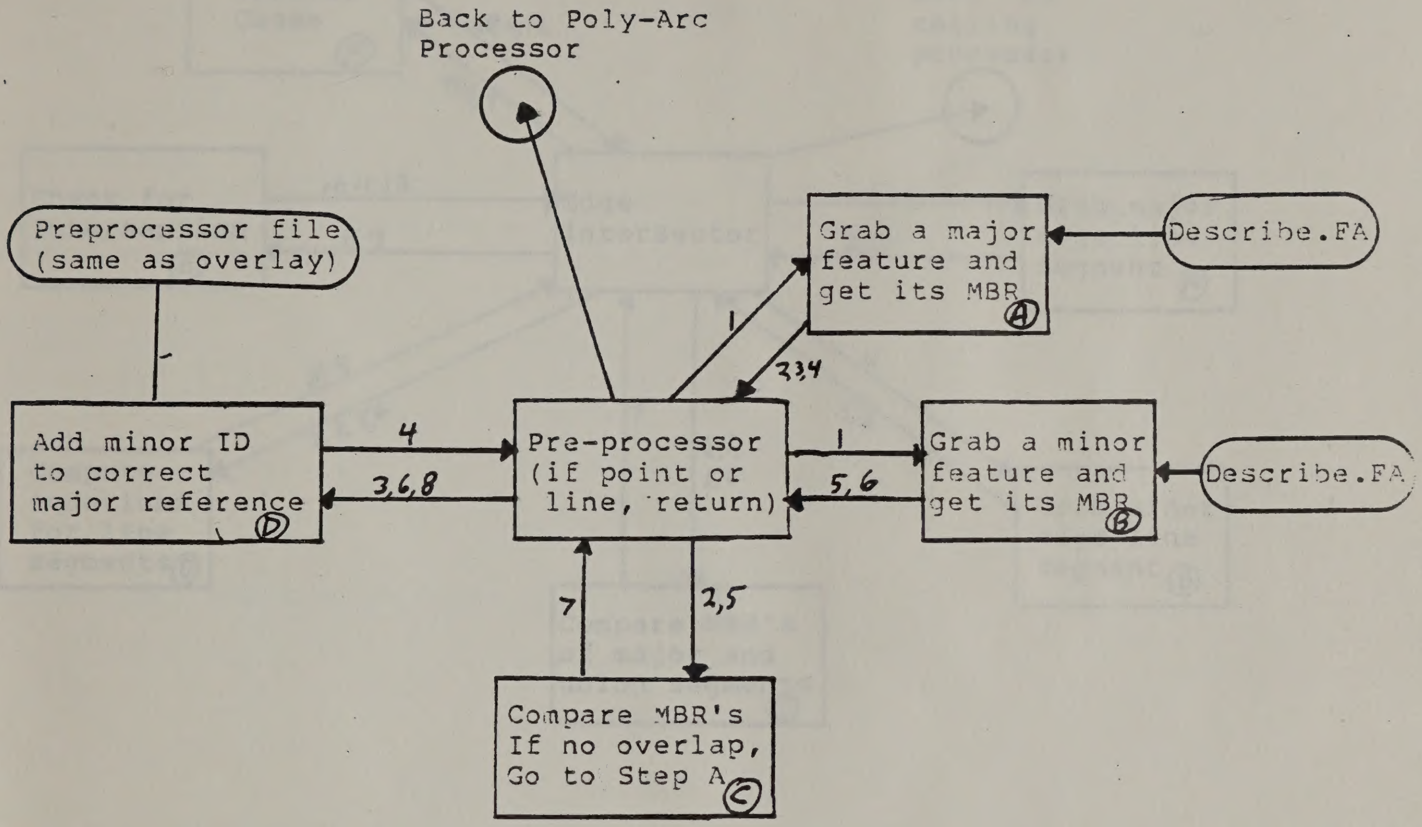

Back to Poly-Arc Processor
Preprocessor file (same as overlay)
Grab a major feature and get its MBR Ⓐ
Describe.FA
1
3,3,4
Add minor ID to correct major reference Ⓓ
4
3,6,8
Pre-processor (if point or line, return)
1
5,6
Grab a minor feature and get its MBR Ⓑ
Describe.FA
7
2,5
Compare MBR's If no overlap, Go to Step A Ⓒ

Figure 3.22

Edge Intersection

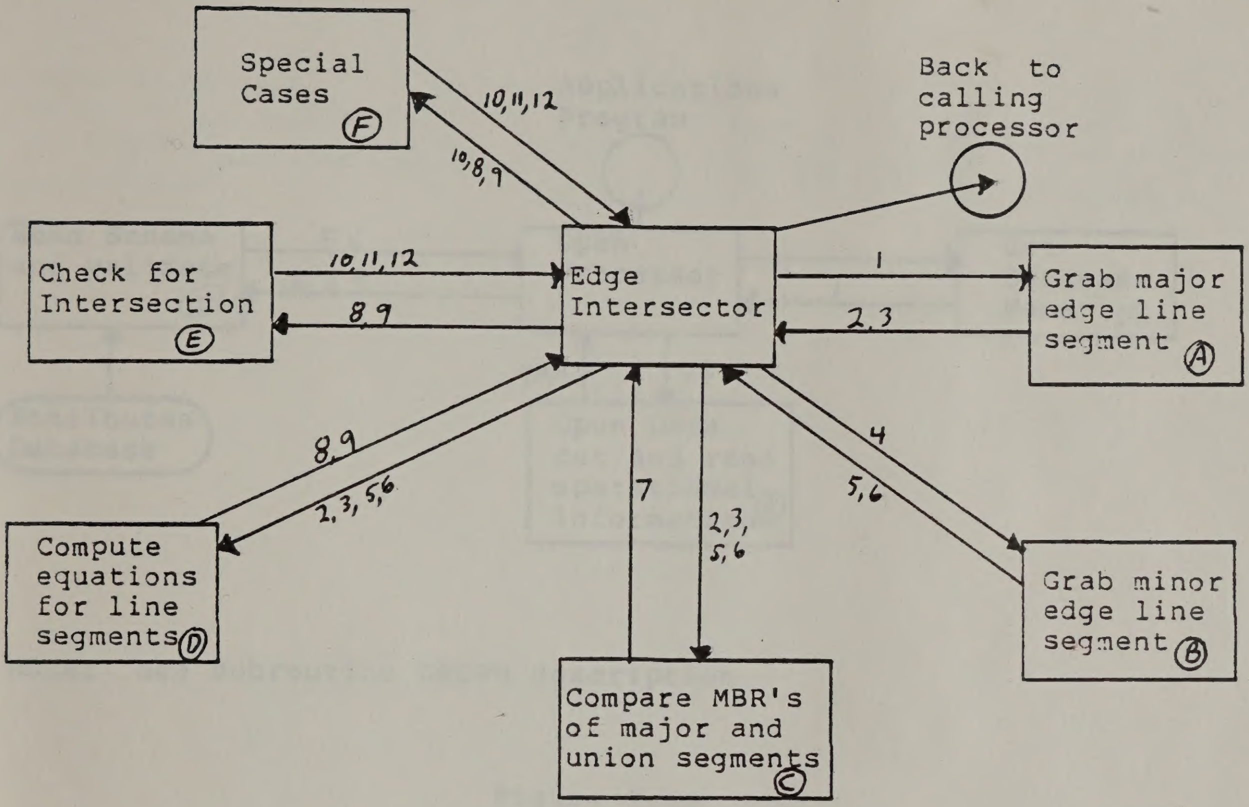
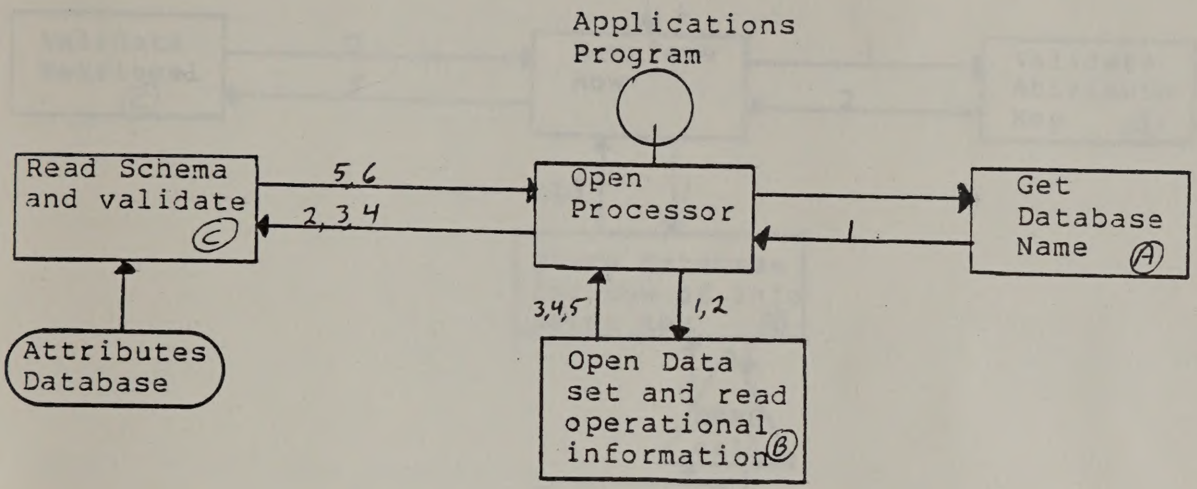

Special Cases
Ⓕ
10,11,12
10,8,9
Back to calling processor
Check for Intersection
Ⓔ
10,11,12
8,9
Edge Intersector
1
2,3
Grab major edge line segment
Ⓐ
8,9
2,3,5,6
7
2,3,5,6
4
5,6
Compute equations for line segments
Ⓓ
Grab minor edge line segment
Ⓑ
Compare MBR's of major and union segments
Ⓒ

Figure 3.23

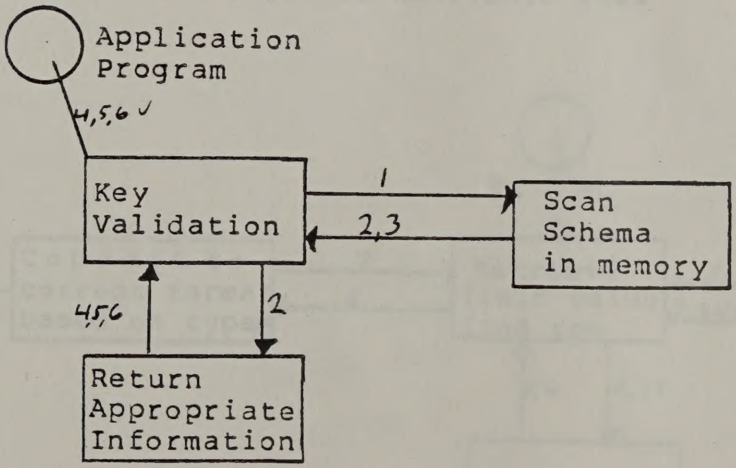
Open Attributes Database


Applications Program
Read Schema and validate
Ⓒ
5,6
2,3,4
Open Processor
Get Database Name
Ⓐ
1
3,4,5
1,2
Attributes Database
Open Data set and read operational information
Ⓑ

NOTE: See subroutine DBOPM description

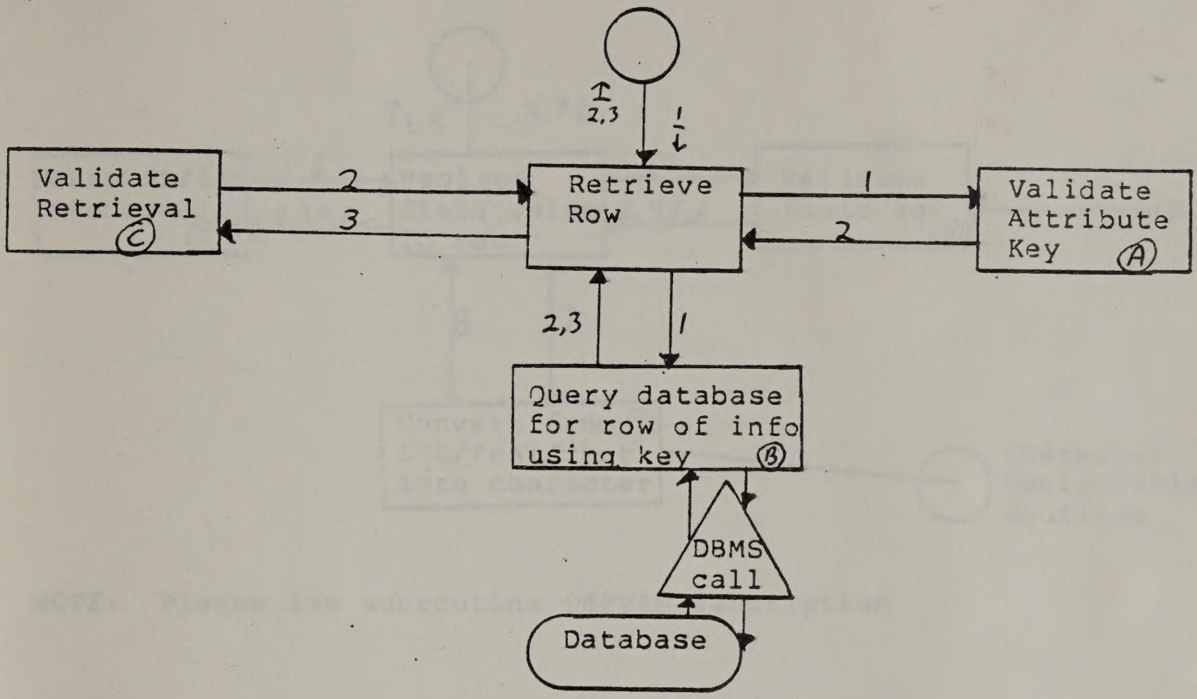
Figure 3.24

Find and Validate Key


Application Program
4,5,6 ✓
Key Validation
1
2,3
Scan Schema in memory
4,5,6
2
Return Appropriate Information

NOTE: See subroutine CHKKEY description

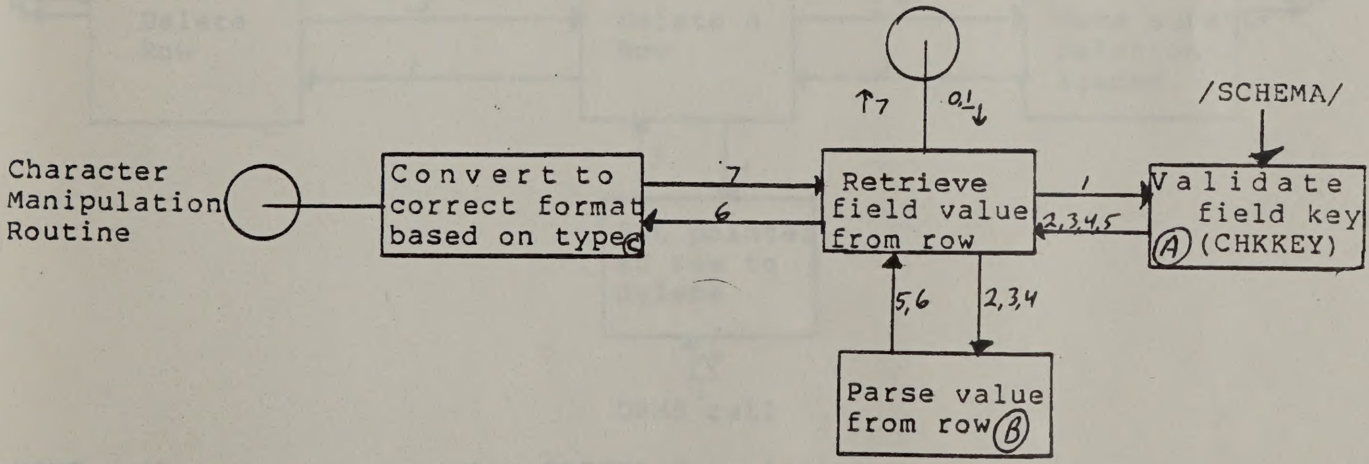
Figure 3.25
Retrieve a
Row from a Table


Validate Retrieval ©
2
3
Retrieve Row
2,3
1
Validate Attribute Key (A)
Query database for row of info using key (B)
DBMS call
Database

NOTE: This is subroutine DBGROW

Figure 3.26

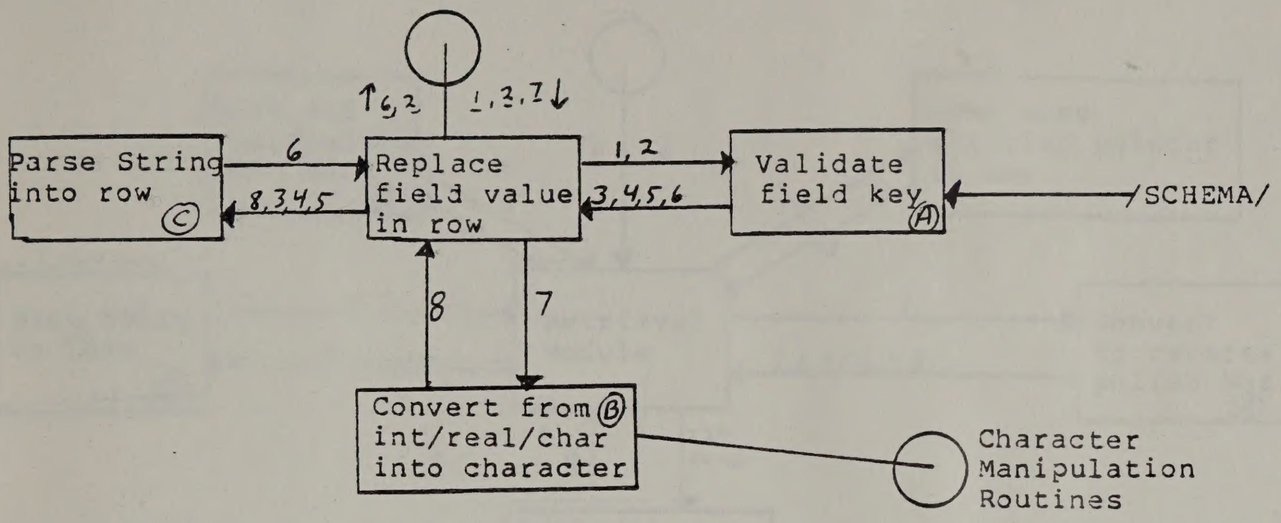
Retrieve a value from a row of attribute data


Character Manipulation Routine
Convert to correct format based on type ©
7
6
Retrieve field value from row
↑7
0,1↓
/SCHEMA/
1
2,3,4,5
Validate field key (A) (CHKKEY)
5,6
2,3,4
Parse value from row (B)

NOTE: Please see subroutine DBGVAR description

Figure 3.27

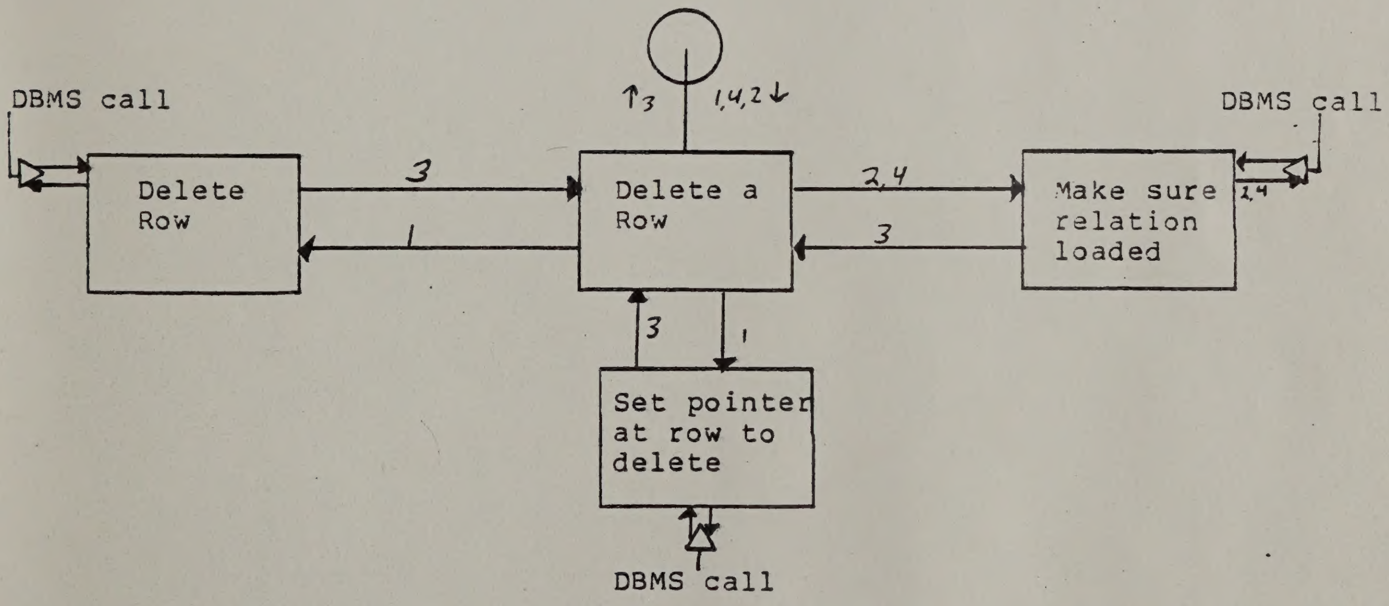
Put value into field location in Row of Attribute Data


↑6,2
1,3,7↓
Parse String into row
Ⓒ
6
8,3,4,5
Replace field value in row
1,2
3,4,5,6
Validate field key
Ⓐ
/SCHEMA/
8
7
Convert from int/real/char into character
Ⓑ
Character Manipulation Routines

NOTE: Please see subroutine DBPVAR description

Figure 3.28

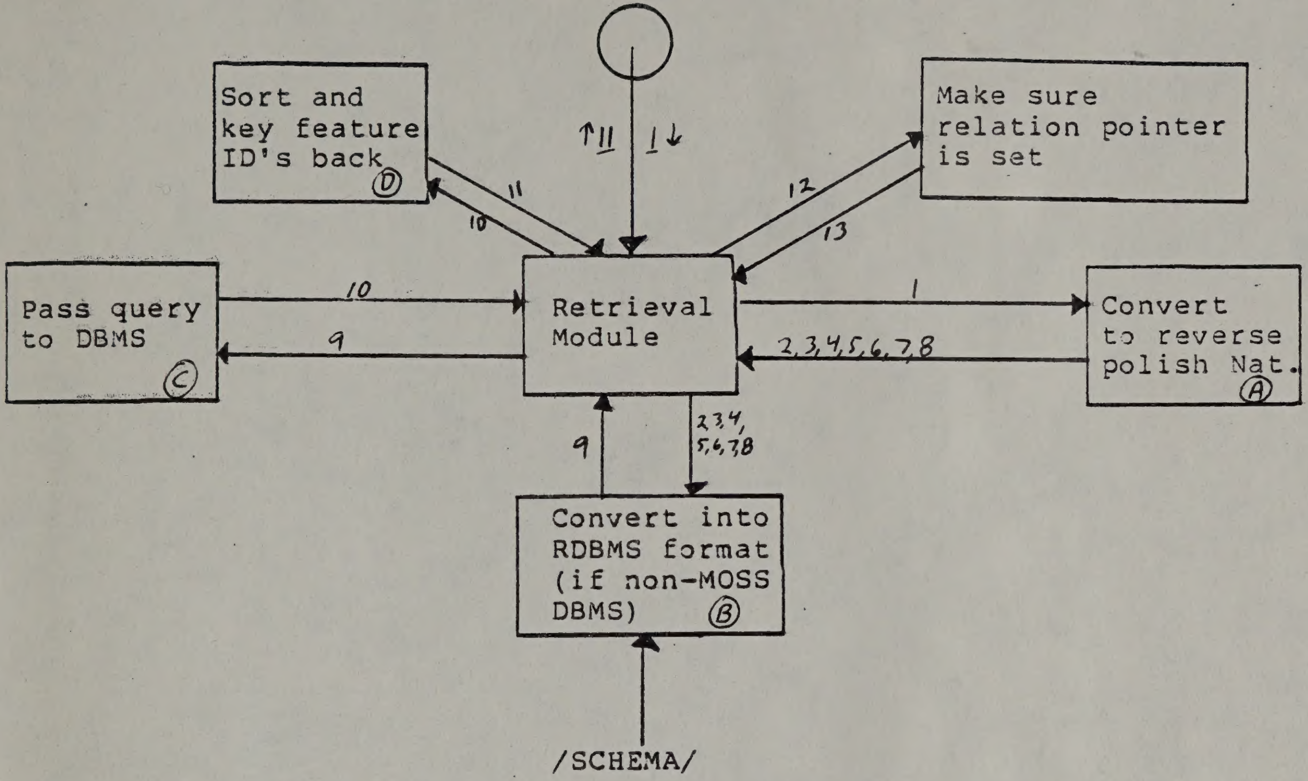
Delete a Row from Attribute File


↑3
1,4,2↓
DBMS call
Delete Row
3
1
Delete a Row
2,4
3
Make sure relation loaded
2,4
DBMS call
3
1
Set pointer at row to delete
DBMS call

NOTE: Please see sunroutine DBRDEL description

Figure 3.29

Attribute Retrieval


Sort and key feature ID's back
Ⓓ
↑11
1↓
Make sure relation pointer is set
11
10
12
13
Pass query to DBMS
Ⓒ
10
9
Retrieval Module
1
2,3,4,5,6,7,8
Convert to reverse polish Nat.
Ⓐ
9
2,3,4, 5,6,7,8
Convert into RDBMS format (if non-MOSS DBMS)
Ⓑ
/SCHEMA/